



Microsoft SharePoint 2010 on AWS: Advanced Implementation Guide

Ulf Schoo

October 2012

(Please consult <http://aws.amazon.com/windows/sharepoint/> for the latest version of this article)

Abstract

Amazon Web Services (AWS) provides a comprehensive set of services and tools for deploying Microsoft Windows-based workloads, including Microsoft SharePoint, on its highly reliable and secure cloud infrastructure. Deploying an enterprise-class SharePoint solution that involves multiple components can be resource and time consuming. We published an article – [Deploy a Microsoft SharePoint 2010 Server Farm in the AWS Cloud in 6 Simple Steps](#) – which includes AWS CloudFormation sample templates so you can launch a fully functional SharePoint 2010 server farm on AWS. The article and templates will help you deploy a public website scenario using SharePoint 2010. The article and this advanced implementation guide build on the [Microsoft SharePoint Server on AWS Reference Architecture](#) whitepaper so you can customize your deployment as necessary.

This guide targets IT infrastructure administrators and DevOps personnel. After reading it, you should have a good understanding about how you can script the deployment and basic configuration process and deploy a SharePoint 2010 server farm on the AWS cloud repeatedly and reliably.

Introduction

SharePoint is a widely deployed application platform, common in many organizations as a platform for public-facing sites and main portal for team–corporate collaboration, content management, workflow, and access to corporate applications. AWS cloud not only provides the on-demand resources (compute, database, network etc.) you need to run this solution but also provides a way to script the provisioning and configuration steps so you can deploy it easily. [AWS CloudFormation](#) enables you to create and provision AWS infrastructure deployments predictably and repeatedly. It helps you deploy AWS services such as Amazon Elastic Compute Cloud (EC2), Amazon Elastic Block Store (EBS), Elastic Load Balancing, and Auto Scaling groups to build reliable, scalable, and cost-efficient applications. In addition, we provide some basic Windows PowerShell scripts for a more detailed configuration of the Windows-based Amazon EC2 instances. These Windows PowerShell scripts provide limited functionality and are not meant to represent a final solution. The scripts are built from samples freely available on the usual Windows PowerShell community sites and are meant to show how you can use AWS CloudFormation and Windows PowerShell to reach deep into your instances at provisioning time and perform the necessary configuration steps. You will want to replace the scripts with your own.

NOTE: The scenario discussed in this guide is that of a public website. We don't discuss the intranet scenario as described in the [Microsoft SharePoint Server on AWS Reference Architecture](#) whitepaper. The accompanying templates, scripts, and methods discussed in this guide serve as a starting point that the reader will later modify or extend. The scripted deployment stops at the stage where the SharePoint 2010 bits are successfully installed on the instances. We don't venture in this article into discussing scripted configuration of SharePoint 2010, as there are no two SharePoint configurations alike. In Step 6, however, we provide a set of detailed instructions that help you configure a SharePoint 2010 team site with full functionality for either a demonstration or proof of concept (POC).

Implementing SharePoint 2010 Server Architecture Scenarios in AWS

This advanced implementation guide provides a walkthrough of the sample templates and describes the AWS-specific implementation details so you can customize them and deploy a solution that best meets your business, IT, and security requirements. This guide follows the outline of the [Deploy a Microsoft SharePoint 2010 Server Farm in the AWS Cloud in 6 Simple Steps](#) article so you can follow along as you launch the sample templates. However, you may also find chapter 2 and 3 useful as a general reference for how to deploy Windows-based infrastructure components such as Microsoft Active Directory and Microsoft SQL Server in the AWS cloud. This guide discusses the following topics:

- **Step 1: Sign up for an AWS Account**
- **Step 2: Launch the virtual Network and Active Directory infrastructure. This includes:**
 - Setting up the virtual network for the multi-tiered SharePoint 2010 server farm within AWS, including subnets in two Availability Zones to support logical server groups for different tiers and roles within the SharePoint reference architecture.
 - Deploying Active Directory to provide authentication and DNS services for the SharePoint 2010 server farm.
 - Configuring Windows Server instances as Remote Desktop Gateways (RD Gateway) to enable secure administrative access, and deploying NAT instances to enable secure communication (e.g., to obtain security and general updates from Windows Update).
 - Security. This section covers implementing security mechanisms in AWS, including how to configure instance and network security to enable authorized access to the overall SharePoint 2010 server farm as well as access between tiers and instances within the farm.
- **Step 3: Launch the Database tier. This includes:**
 - Creating an AWS CloudFormation-enabled SQL Server 2008 R2 Standard Edition Amazon Machine Image (AMI) to enable scripted configuration of the SQL Server components of the farm.
 - Joining the SQL Server instance to the domain.
 - Provisioning of SharePoint farm administrator logins and *dbcreator* and *securityadmin* roles using Windows PowerShell.
- **Step 4: Launch the Application-Server tier. This includes:**
 - Creating an extended Windows Server AMI that holds the raw (uninstalled) SharePoint bits and the Windows PowerShell module SPModule to reduce deployment time.
 - Installing SharePoint application servers (using the [License Mobility through Software Assurance model](#)) using the SharePoint deployment Windows PowerShell scripts.
- **Step 5: Launch the Web Front-end (WFE) tier. This includes:**
 - Installing WFE servers (one per Availability Zone) to enable load-balanced access to the SharePoint Web application, using the SharePoint AMI created in the previous step.
 - Deploying Amazon Elastic Load Balancer in front of the WFEs.
- **Step 6: Configure the SharePoint Farm Servers. This includes:**
 - Configuring application-tier server instances using the SharePoint Products and Technologies Configuration Wizard.
 - Configuring WFE server instances using the SharePoint Products and Technologies Configuration Wizard.
 - Testing your SharePoint 2010 deployment and demonstrating the facilities of the default team site.

When complete, your SharePoint 2010 server farm implements the following scenario:

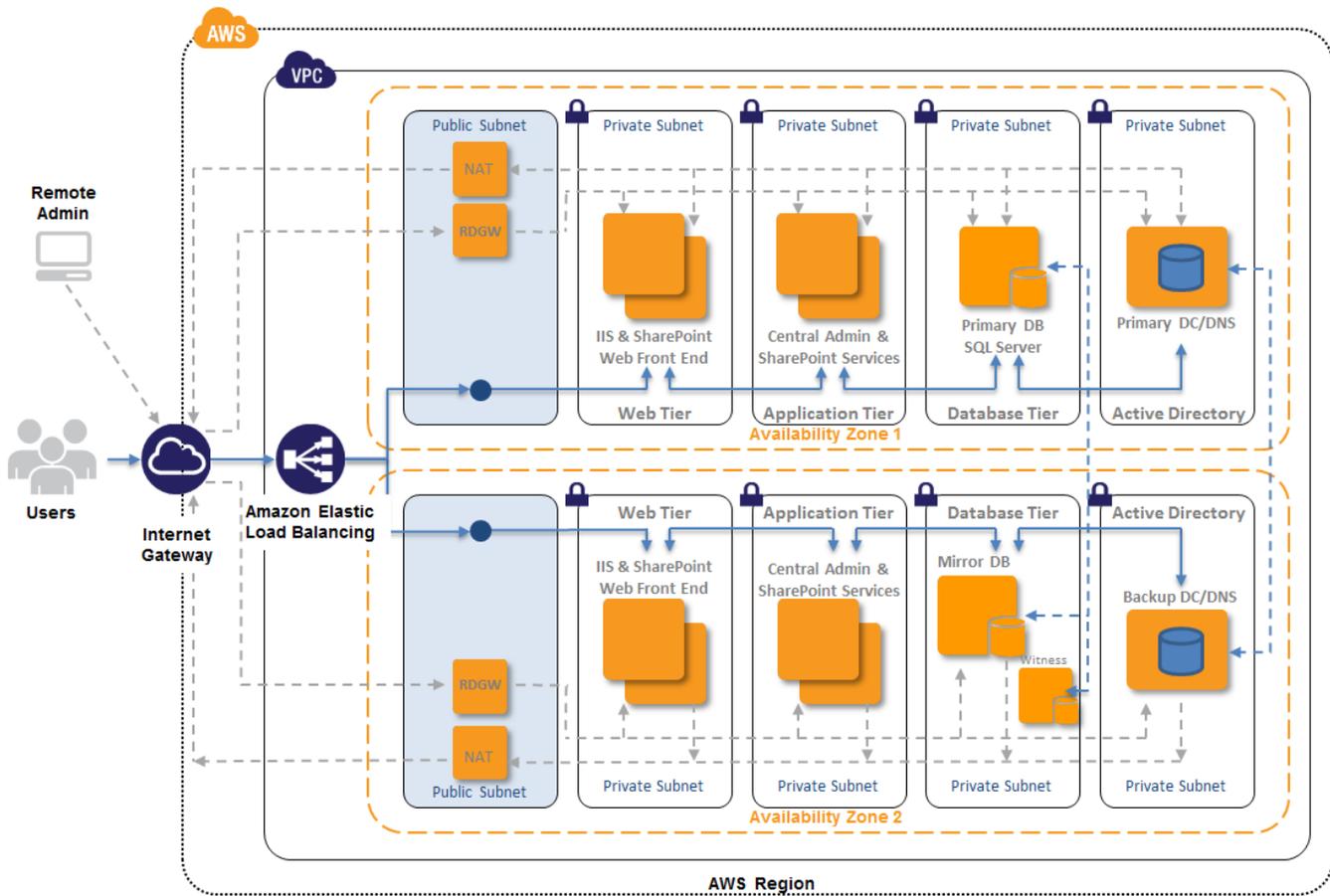


Figure 1: SharePoint server reference architecture for the public website scenario

Step 1: Sign up for an AWS Account

If you already have an AWS account, skip to the next step. If you don't already have an AWS account, use the following procedure to create one.

To create an AWS account, go to <http://aws.amazon.com>, and click **Sign Up Now**. Follow the on-screen instructions. Part of the sign-up process involves receiving a phone call and entering a PIN using the phone keypad.

When you create an AWS account, AWS automatically signs up the account for all AWS services, including Amazon EC2. You are charged only for the services that you use.

Step 2: Launch the virtual Network and Active Directory infrastructure

Let's start with the necessary infrastructure and virtual network setup to provide the environment in which you instantiate and configure your servers and database.

The [Microsoft SharePoint Server on AWS Reference Architecture](#) whitepaper is organized around a multi-tiered (web, application, and database) approach, allowing you to independently scale and configure each tier. Your first task is to define a virtual network environment that supports this type of tiered structure and enables you to deploy the various server roles in each tier with suitable security configuration.

Note: The public website scenario of the [Microsoft SharePoint Server on AWS Reference Architecture](#) is deployed into an Amazon Virtual Private Cloud (Amazon VPC). Amazon VPC lets you provision a private, isolated section of the AWS cloud where you can launch AWS resources in a virtual network that you define. With Amazon VPC, you can define a virtual network topology closely resembling a traditional network that you might operate in your own datacenter. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways.

Setting Up Amazon VPC for the Public Website SharePoint Server Farm Scenario

For the public website scenario, we are accommodating the following requirements:

- We want to launch the web, application, and database tiers in private subnets; users only need to get to the load balancers (which are deployed in Part 3 after the WFE instances are created).
- It is advisable for the public website scenario to deploy additional components at the front end for firewall and threat management. It is out of scope for this article and the accompanying artifacts to automatically deploy a product such as [Microsoft Forefront Threat Management Gateway](#). However, you might want to consider deploying such a product or products with comparable functionality manually after all the scripts have been run.
- We add NAT instances in each Availability Zone to facilitate servers in private subnets communicating out to the Internet (to receive operating system software updates, for example).

Given the preceding requirements, Figure 2 shows the network setup and administrative access for the public website scenario:

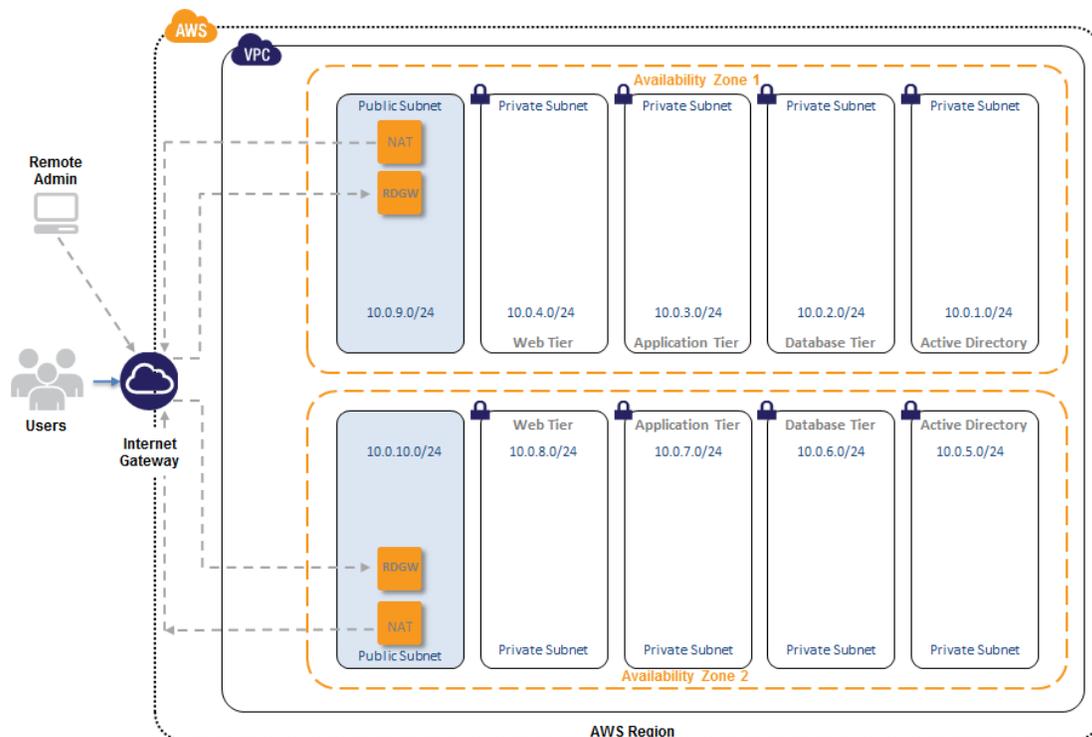


Figure 2: Network configuration and administrative access for the Internet-facing public website scenario

Using Sample Template-1

You might want to open up the sample [Template-1](#) AWS CloudFormation template file and follow along.

Template Customization

Sample Template-1 allows for rich customization of 30 defined parameters at template launch. You can modify those parameters, change the default values, or create an entirely new set of parameters based on your specific deployment scenario. AWS CloudFormation currently supports a maximum of 30 parameters per template. The Template-1 parameters include the following default values.

Parameter	Default	Description
KeyPairName	<User Provides>	Public/private key pairs allow you to connect securely to your instance after it launches.
ADInstanceType	m1.xlarge	Amazon EC2 instance type for the first Active Directory instance.
AD2InstanceType	m1.xlarge	Amazon EC2 instance type for the second Active Directory instance.
NATInstanceType	m1.small	Amazon EC2 instance type for the NAT instances.
RDGWInstanceType	m1.large	Amazon EC2 instance type for the Remote Desktop Gateway instances.
DomainDNSName	contoso.com	Fully qualified domain name (FQDN) of the forest root domain; e.g., corp.example.com.
DomainNetBIOSName	contoso	NetBIOS name of the domain (up to 15 characters) for users of earlier versions of Windows; e.g., CORP.
ServerNetBIOSName1	DC1	NetBIOS name of the first Active Directory server (up to 15 characters).
ServerNetBIOSName2	DC2	NetBIOS name of the second Active Directory server (up to 15 characters).
RestoreModePassword	Password123	Password for a separate administrator account when the domain controller is in restore mode. Must be at least 8 characters containing letters, numbers, and symbols.
DomainAdminUser	StackAdmin	User name for the account that is added as domain administrator. This is separate from the default "administrator" account.
DomainAdminPassword	Password123	Password for the domain admin user. Must be at least 8 characters containing letters, numbers, and symbols.
SPSAdminUser	SPFarmAdmin	User name for the SharePoint server admin account. This account is a domain user and is added to the SQL Server database as a member of the <i>dbcreator</i> role.
SPSAdminPassword	Password123	Password for the SPS admin user. Must be at least 8 characters containing letters, numbers, and symbols.
UserCSVSourceLocation	https://s3.amazonaws.com/CFN_Templates/UserImport.csv	An accessible source location of a comma separated values (.CSV) file containing any user sAMAccountName and the name you may want to pre-provision Active Directory with.
AZ1	us-east-1a	Name of Availability Zone that will contain public and private subnets; select a valid zone for your region.
AZ2	us-east-1b	Name of Availability Zone that will contain public and private subnets; select a valid zone for your region.
DMZ1CIDR	10.0.9.0/24	CIDR Block for the Public DMZ subnet located in AZ1
DMZ2CIDR	10.0.10.0/24	CIDR Block for the Public DMZ subnet located in AZ2
PrivSub1CIDR	10.0.1.0/24	CIDR Block for Private Subnet 1 located in AZ1
PrivSub2CIDR	10.0.2.0/24	CIDR Block for Private Subnet 2 located in AZ1
PrivSub3CIDR	10.0.3.0/24	CIDR Block for Private Subnet 3 located in AZ1
PrivSub4CIDR	10.0.4.0/24	CIDR Block for Private Subnet 4 located in AZ1
PrivSub5CIDR	10.0.5.0/24	CIDR Block for Private Subnet 5 located in AZ2
PrivSub6CIDR	10.0.6.0/24	CIDR Block for Private Subnet 6 located in AZ2
PrivSub7CIDR	10.0.7.0/24	CIDR Block for Private Subnet 7 located in AZ2
PrivSub8CIDR	10.0.8.0/24	CIDR Block for Private Subnet 8 located in AZ2
VPCCIDR	10.0.0.0/16	CIDR Block for the VPC
AD1PrivateIp	10.0.1.10	Fixed private IP for the first Active Directory server located in AZ1
AD2PrivateIp	10.0.5.10	Fixed private IP for the second Active Directory server located in AZ2

Figure 3: Template-1 parameters

VPC and Subnet Setup

Creating a VPC using AWS CloudFormation requires only a few lines of code in the Resources section of your template. This launches a resource of the type [AWS::EC2::VPC](#).

```
"VPC" : {
  "Type" : "AWS::EC2::VPC",
  "Properties" : {
    "CidrBlock" : { "Ref" : "VPCCIDR" },
    "Tags" : [
      { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
      { "Key" : "Network", "Value" : "Public" }
    ]
  }
},
```

As in Figure 3, we want to give the users of our templates control over the definition of the CIDR block for the VPC. To do so, we need to declare a parameter in the Parameters section of our template that we can then reference { "Ref" : "VPCCIDR" } when creating the VPC resource itself or resources associated with this VPC. This parameter definition is as follows:

```
"VPCCIDR" : {
  "Description" : "CIDR Block for the VPC",
  "Type" : "String",
  "Default" : "10.0.0.0/16",
  "AllowedPattern" : "[a-zA-Z0-9]+\.\.\."
},
```

Next, we create the eight private subnets and we do this by following a similar pattern as we used for creating the VPC. First, we declare a resource of the type [AWS::EC2::Subnet](#):

```
"PrivateSubnet1" : {
  "Type" : "AWS::EC2::Subnet",
  "Properties" : {
    "VpcId" : { "Ref" : "VPC" },
    "CidrBlock" : { "Ref" : "PrivSub1CIDR" },
    "AvailabilityZone" : { "Ref" : "AZ1" },
    "Tags" : [
      { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
      { "Key" : "Network", "Value" : "Private" },
      { "Key" : "Role", "Value" : "AD1 Subnet" }
    ]
  }
},
```

We are using references to four different types of resources.

- { "Ref" : "VPC" } is a reference to the VPC created in the previous step. Launch all subnets into this VPC.
- { "Ref" : "PrivSub1CIDR" } is a reference to the CIDR block for this private subnet as we want to give users of the template the ability to define the IP ranges for each subnet to best match what they are used to from their on-premise deployment. This parameter definition is as follows:

```
"PrivSub1CIDR" : {
  "Description" : "CIDR Block for Private Subnet 1 located in AZ1",
  "Type" : "String",
  "Default" : "10.0.1.0/24",
  "AllowedPattern" : "[a-zA-Z0-9]+\.\.\."
},
```

- { "Ref" : "AZ1" } is a reference to the Availability Zone in which you want to create the subnet. As we outlined earlier, we want to set up a mirror in two Availability Zones to provide redundancy and failover. This parameter definition for AZ1 is as follows (the definition is similar for AZ2):

```
"AZ1" : {
  "Description" : "Name of Availability Zone that will contain public & private subnets - Select a valid zone
    for your region",
  "Type" : "String",
  "Default" : "us-east-1a",
  "AllowedValues" : ["eu-west-1a","eu-west-1b","eu-west-1c","us-east-1a","us-east-
    1b","us-east-1c","us-east-
    1d","us-west-1a","us-west-1b"],
  "ConstraintDescription" : "must be a valid EC2 Availability Zone for region being
    deployed to. Only supports eu-
    west-1 ,us-east-1 & us-west-1 <- You can expand"
},
```

- We are using a reference to the StackName property { "Ref" : "AWS::StackName"} to tag our subnet.

Besides the eight private subnets, we also want to deploy two public subnets. Deploying public subnets follows the same pattern as described earlier with the private subnets. The only two things that distinguish public subnets from private are: the route (e.g., the public route channels Internet traffic directly to the Internet gateway while the private route channels Internet traffic to the NAT instance); and that instances in the public subnet actually have an Internet-routable IP address. We discuss how to define and encode public and private routes in the next section.

Private and Public Routes

After we create the VPC and the subnets inside the VPC, we need to define how traffic will flow inside the VPC and out of the VPC. We define the routes: one route for defining the traffic flow for all the private subnets, and one route for the two public subnets.

Before we can create those routes, however, we need to define the means by which the VPC communicates with the Internet. We create an Internet gateway resource of the type [AWS::EC2::InternetGateway](#) and by launching a NAT instance, with few lines of script. This script is as follows:

```
"InternetGateway" : {
  "Type" : "AWS::EC2::InternetGateway",
  "Properties" : {
    "Tags" : [
      { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
      { "Key" : "Network", "Value" : "Public" }
    ]
  }
},
```

After we create the Internet gateway, we only have to attach the gateway to the VPC. The code for doing this is as follows:

```
"AttachGateway" : {
  "Type" : "AWS::EC2::VPCGatewayAttachment",
  "Properties" : {
    "VpcId" : { "Ref" : "VPC" },
    "InternetGatewayId" : { "Ref" : "InternetGateway" }
  }
},
```

Next, we create the NAT instance in each Availability Zone to facilitate servers in private subnets communicating out to the Internet (to get operating system software updates, for example). The code for doing this is as follows:

```
"NAT1" : {
  "Type" : "AWS::EC2::Instance",
  "Properties" : {
    "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArchNatAMI", { "Ref" :
      "AWS::Region" }, { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref"
        : "NATInstanceType" }, "Arch" ] } ] },
    "InstanceType" : { "Ref" : "NATInstanceType" },
    "SubnetId" : { "Ref" : "DMZSubnet" },
    "Tags" : [ {
      "Key" : "Name",
      "Value" : "NAT1"
    } ],
    "SecurityGroupIds" : [ { "Ref" : "NAT1SecurityGroup" } ],
    "KeyName" : { "Ref" : "KeyPairName" },
    "SourceDestCheck" : "false"
  }
},
```

Similar to other VPC and subnet resources, we are extensively using references either to previously-created resources like the DMZ Subnet { "Ref" : "DMZSubnet" } that we want to launch this instance into, or to a security group that governs the type of traffic we allow to flow in or out of this instance. (We discuss security group setup in detail later in this article.)

As the NAT instance resides in a public subnet, it also needs a publicly routable IP address. We achieve this by creating an EIP resource { "Type" : "[AWS::EC2::EIP](#)" } and associating it with the instance. The code for doing this is as follows in AWS CloudFormation:

```
"NAT1EIP" : {
  "Type" : "AWS::EC2::EIP",
  "Properties" : {
    "Domain" : "vpc",
    "InstanceId" : { "Ref" : "NAT1" }
  }
},
```

Now that we have both our Internet gateway and NAT instance deployed, we can construct our routes and associate the routes with the proper subnet.

First, we create the private route table. This looks as follows:

```
"PrivateRouteTable" : {
  "Type" : "AWS::EC2::RouteTable",
  "Properties" : {
    "VpcId" : { "Ref" : "VPC" },
    "Tags" : [
      { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
      { "Key" : "Network", "Value" : "AZ1 Private" }
    ]
  }
},
```

Then we construct the private route that is associated to the route table.

```
"PrivateRoute" : {
  "Type" : "AWS::EC2::Route",
  "Properties" : {
    "RouteTableId" : { "Ref" : "PrivateRouteTable" },
    "DestinationCidrBlock" : "0.0.0.0/0",
    "InstanceId" : { "Ref" : "NAT1" }
  }
},
```

```
    }
  },
```

What this route defines is that all traffic destined for the Internet ("DestinationCidrBlock" : "0.0.0.0/0") has to go through the NAT instance we created earlier. Now we need to associate all eight of our private subnets with this route, and the code for doing this is as follows:

```
"PrivateSubnetRouteTableAssociation1" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "PrivateSubnet1" },
    "RouteTableId" : { "Ref" : "PrivateRouteTable" }
  }
},
```

This takes care of all outbound traffic from our private subnets to the Internet. What about traffic from the Internet to other types on instances we will deploy at a later stage into the DMZ, like an RD Gateway for securely logging into your instances? This traffic is routed via our public route and it follows the same pattern as established with the private route. First, we create the DMZ (public) route table as follows:

```
"DMZRouteTable" : {
  "Type" : "AWS::EC2::RouteTable",
  "Properties" : {
    "VpcId" : { "Ref" : "VPC" },
    "Tags" : [
      { "Key" : "Application", "Value" : { "Ref" : "AWS::StackName" } },
      { "Key" : "Network", "Value" : "DMZ" }
    ]
  }
},
```

Then we construct the DMZ (public) route and associate it with the DMZ route table. This is why we first had to create the Internet gateway resource (and the NAT instance earlier).

```
"DMZRoute" : {
  "Type" : "AWS::EC2::Route",
  "Properties" : {
    "RouteTableId" : { "Ref" : "DMZRouteTable" },
    "DestinationCidrBlock" : "0.0.0.0/0",
    "GatewayId" : { "Ref" : "InternetGateway" }
  }
},
```

This route defines that all traffic destined for the Internet ("DestinationCidrBlock" : "0.0.0.0/0") has to go through the Internet gateway we created earlier. Now we need to associate the two public (DMZ) subnets with this route and the code for doing this is as follows:

```
"DMZ1SubnetRouteTableAssociation" : {
  "Type" : "AWS::EC2::SubnetRouteTableAssociation",
  "Properties" : {
    "SubnetId" : { "Ref" : "DMZSubnet" },
    "RouteTableId" : { "Ref" : "DMZRouteTable" }
  }
},
```

Similar to the private routes, we want to have our network setup mirrored in a second Availability Zone.

After this step, we are ready to deploy the Active Directory instances into our VPC. If we try to deploy an Active Directory instance in a private subnet inside a VPC without the proper public and private routing setup, the instance itself is created but none of the subsequent configuration scripts are executed. AWS CloudFormation requires access to

the [CloudFormation Endpoints](#) where, among other things we discuss later, state of Wait Conditions is stored so we can orchestrate the right launch order of our resources.

AD DS Setup and DNS Configuration

SharePoint 2010 requires Active Directory Domain Services (AD DS) for user authentication and availability of the complete feature set. However, you can also leverage AD DS to provide Domain Name Server (DNS) functionality within the VPC among the various server instances.

For your SharePoint server farm to operate, you need connectivity to one or more domain controllers to facilitate user authentication and DNS resolution across servers within the farm.

NOTE: It is also possible to support this scenario for corporate environments that do not use AD DS but rather another Lightweight Directory Access Protocol (LDAP)–based directory service. You can use [Active Directory Federation Services \(AD FS\)](#) with SharePoint and other, non–AD DS authentication providers to facilitate federated authentication. AWS provides a detailed whitepaper, [Step by Step: Single Sign-On to Amazon EC2-Based .NET Applications from an On-Premises Windows Domain](#), on how to set up and configure AD FS in AWS to support federated authentication.

In our public website scenario, the SharePoint server farm does not use VPN to connect to a corporate infrastructure. Instead, it requires AD DS to be instantiated within the AWS environment to facilitate user registration and authentication for the SharePoint instances running there. For more information on detailed setup and configuration steps, see the “Windows Server Setup and Configuration” section. We suggest hosting domain controllers in multiple Availability Zones to provide redundancy and high availability, as illustrated in Figure 4.

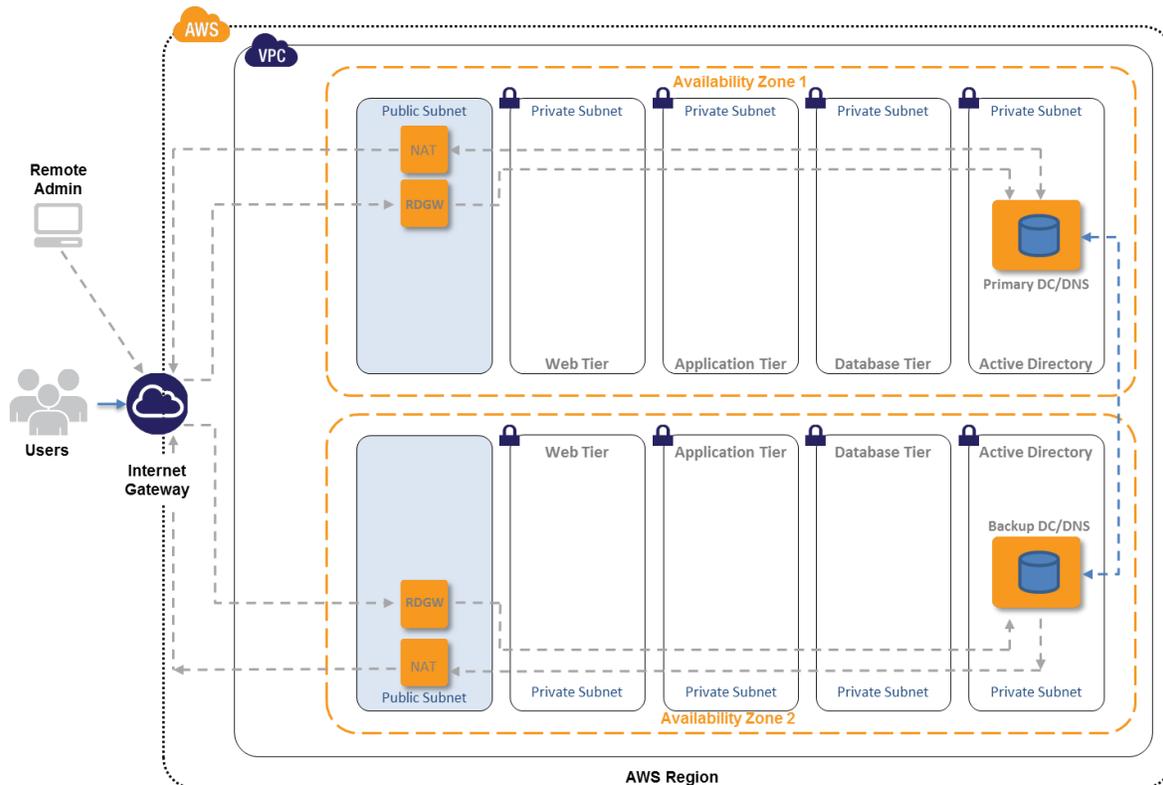


Figure 4: Hosting domain controllers in multiple Availability Zones to provide redundancy and high availability

Launching your stack in the right order

So, how do we set all this up via an AWS CloudFormation template? Before we begin, let us briefly discuss in a little more detail how we launch in the right order, as this is equally important. As we already mentioned towards the end of the previous section, AWS deployment and configuration steps must be executed in the right order. AWS CloudFormation provides for this via the [AWS::CloudFormation::WaitCondition](#) and [AWS::CloudFormation::WaitConditionHandle](#) resources.

To make this more concrete, for our scenario we have to take a dependency on both NAT instances being created successfully before we can launch the Windows instance that we subsequently configure to be our domain controller. We also have to wait for the primary domain controller to be fully deployed before we can launch the second Windows instance that we subsequently also promote to be a domain controller in the second Availability Zone.

The order is as follows: NAT1 launches first. For NAT2, we specify a dependency on NAT1 (which is not technically accurate but is necessary if we want to control the right order). For simple NAT instances, there is no penalty for specifying a dependency. Our first domain controller (DC1) has a dependency on NAT2 being launched. Domain Controller 2 (DC2) launches only after DC1 configured fully with the intended server roles and properly provisioned. Our RD Gateway, which serves as a bastion host or jump server to administrate the instances in the private subnet securely, then needs to follow. To summarize, the order is as follows:

NAT1 → NAT2 → DC1 → DC2 → RDGW1 → RDGW2

A simple "DependsOn": "NAT1" statement is sufficient if, as in the case of the NAT instances, the dependency is only on the launch of the instance without any additional configuration steps. However, if we are performing longer-running configuration tasks on an instance and all tasks need to complete successfully before we can declare the instance running and properly configured, we need to construct a WaitCondition and associated WaitHandle resource. In AWS CloudFormation, this code is as follows for the domain controller:

```
"DomainControllerWaitCondition" : {
  "Type" : "AWS::CloudFormation::WaitCondition",
  "DependsOn" : "DomainController",
  "Properties" : {
    "Handle" : {"Ref" : "DomainControllerWaitHandle"},
    "Timeout" : "1800"
  }
},
"DomainControllerWaitHandle" : {
  "Type" : "AWS::CloudFormation::WaitConditionHandle"
},
```

You should adjust the **Timeout** property as needed, taking into account your specific configuration. The value 1800 specifies a 30-minute timeout period (1800/60 seconds) and is sufficient for the configuration steps we perform in our sample template.

The very last step in all the configurations we perform on our Windows-based instance is to signal success (or failure) to the WaitHandle resource and the code is as follows:

```
"3-signal-success" : {
  "command" : { "Fn::Join" : [ "", [
    "cfn-signal.exe -e 0 \"", { "Ref" : "DomainControllerWaitHandle" }, "\" ] ]"
  ] }
}
```

Here we are using one of the four [AWS CloudFormation helper scripts](#), `cfn-signal.exe`, to signal success to the domain controller `WaitHandle` resource we constructed earlier. (For more information regarding AWS CloudFormation helper scripts, consult the [AWS CloudFormation documentation](#).)

Setting up and configuring Windows Server

Now that we have launched our instances in the right order, let's take a closer look at how we configure and provision our Windows Server instance with the AD DS and DNS server role.

In the previous section, we got introduced to one of the AWS CloudFormation helper scripts: `cfn-signal.exe`. The real workhorse of the AWS CloudFormation helper scripts—`cfn-init`—provides us with the ability to execute a number of detailed configuration tasks on our Windows-based instances. The `cfn-init` helper script reads template metadata from the `AWS::CloudFormation::Init` key and acts accordingly to perform the following tasks:

- Fetch and parse metadata from AWS CloudFormation
- Install packages
- Write files to disk
- Enable/disable and start/stop services

For more information about the template metadata that `cfn-init` uses, see [AWS::CloudFormation::Init](#).

So let's understand how to accomplish this in our code.

We invoke the `cfn-init` helper script in the user data section of the properties of your instance; this code is as follows:

```
"UserData" : { "Fn::Base64" : { "Fn::Join" : [ "", [
    "<script>\n",
    "cfn-init.exe -v -c config -s ", { "Ref" : "AWS::StackName" },
    " -r DomainController ",
    "--access-key ", { "Ref" : "IAMUserAccessKey" },
    "--secret-key ", {"Fn::GetAtt": ["IAMUserAccessKey", "SecretAccessKey"]},
    "--region ", { "Ref" : "AWS::Region" }, "\n", "</script>"
  ] } }
```

Let's look at what is going on here so we understand what the `cfn-init` requires, as that determines our next steps. The `cfn-init` script looks for a `configsets` option (`-c`) with the name `config` and the logical resource ID of our domain controller. At this point, you might recall the earlier discussion about why we need to deploy the NAT instances first and have the public and private routes set up to successfully launch an instance that uses AWS CloudFormation and the CloudFormation helper scripts to configure the instance. The `cfn-init` helper script needs access to the Internet and the Amazon Simple Storage Service (S3) to derive, amongst other things, the AWS CloudFormation URL. The `-access-key`, `-secret-key`, and `-region` provided in the user data section are necessary for `cfn-init` to obtain the necessary information securely.

As we just defined, all the action really happens in the configset called "config" which lives in the metadata section on the instance under the `AWS::CloudFormation::Init` key. Let's move on and create building blocks.

```
"DomainController": {
  "Type" : "AWS::EC2::Instance",
  "DependsOn" : "NAT2",
  "Metadata" : {
    "AWS::CloudFormation::Init" : {
```

```

    "configSets" : {
      "config" : ["setup", "rename", "dcpromo", "createsite",
        "finalize"]
    },

```

Our configset contains several building blocks that we named setup, rename, dcpromo, createsite, and finalize. You will find these blocks either partially or completely reused in many of the subsequently discussed scripts.

We are using the “Setup” section to assemble any Windows PowerShell and other file types where we want to use parameters provided from the AWS CloudFormation template in our Windows PowerShell script or other small scripts. A good example for a Windows PowerShell script that uses user-provided input from the AWS CloudFormation template parameters is our `CreateUsers.ps1` file. We have included this in the script to provide a sample for how an administrator might want to pre-provision the Active Directory users from a CSV file so that his/her deployment is ready to go after all instances are deployed and configured. Your specific deployment might not have any need for this. Specifically, this PS1 file allows an administrator to specify a location for a CSV file containing the names, logins, and temporary password for a set of users that need to be pre-provisioned on the newly created domain. While this is certainly a rudimentary sample that must be built out for use in enterprise deployment, it is useful for quickly setting up a demo or training workshop environment. The code to do this is as follows:

```

"C:\\cfn\\scripts\\CreateUsers.ps1" : {
  "content" : { "Fn::Join" : [ "", [
    "import-module ActiveDirectory ", "\n",
    "import-module BitsTransfer ", "\n",
    "Start-BitsTransfer ", "-Source ", { "Ref" :
    "UserCSVSourceLocation" }, " -Destination
    c:\\cfn\\scripts\\UserImport.csv", "\n",
    " Import-CSV c:\\cfn\\scripts\\UserImport.csv", " |
    ", "foreach ", "{New-AdUser -SamAccountName
    $_.SamAccountName -Name $_.Name -AccountPassword
    (ConvertTo-SecureString ", "\"Password123$" ", "-
    AsPlainText -Force) -Enabled $true - PasswordNeverExpires $true -Path ",
    "\"CN=Users,",
    "DC=", { "Ref" : "DomainNetBIOSName" },
    ",DC=Com\""}, "\n"
  ] ] }
},

```

In subsequent sections, such as the rename section, we run commands that either execute the Windows PowerShell script blocks created in the setup section or execute a single command, as in the reboot command. Executing a Windows PowerShell script is as follows:

```

"rename" : {
  "commands" : {
    "1-execute-powershell-script-RenameComputer" : {
      "command" : { "Fn::Join" : [ "", [
        "powershell.exe ", "-ExecutionPolicy", "
        Unrestricted", "
        C:\\cfn\\scripts\\RenameComputer.ps1 ",
        { "Ref" : "ServerNetBIOSName1" }
      ] ] }
    },
    "2-reboot" : {
      "command" : "shutdown -r -t 1"
    }
  }
},

```

Now that we understand how to create files and execute commands on our Windows-based instance, it becomes fairly straightforward to understand how the next configuration steps are encoded and executed. In the `dcpromo` block of our configset, we promote the Windows Server instance to be a domain controller for a new forest and it also serves as

a DNS server that holds the global catalog. Through the AWS CloudFormation template, the user can choose the domain DNS name (FQDN), the domain NetBIOS name, and the safe mode admin password.

After that, we create our domain admin and SharePoint administrator account and we execute the Windows PowerShell script we assembled earlier on to provision a larger batch of Active Directory users.

You will find at the end of each command in our script a line that either says `"waitAfterCompletion" : "forever"` or `"waitAfterCompletion" : "0"`. Why do we add these commands? AWS CloudFormation, by design, pauses for 60 seconds after execution of a command. This allows any reboots to happen before continuing, among other reasons. So, this should take care of the scenario where we force a reboot after the `dcpromo` block. However, sometimes Windows takes longer to actually perform the reboot. For this scenario, when we know a reboot has to happen, we insert `"waitAfterCompletion" : "forever"`. For all other cases, we can speed up the execution of each command by telling AWS CloudFormation to move right on to the next step. We do this by inserting `"waitAfterCompletion" : "0"`.

To highlight further the fine control an administrator or DevOps person has in fully configuring an Active Directory environment, we execute a longer and more complex Windows PowerShell script that creates the physical layout of our Active Directory infrastructure as deployed in the AWS VPC by creating AD sites, subnets, and proper site links.

Note: This script, like all the other scripts provided, is not meant to be the ultimate solution but instead is meant to provide an example for what you can do and how far you can take this when you code your scripts for your deployment. Before rolling this out into a production environment, you might want to consider, for example, providing scripts that configure group policies appropriate for your organization.

Lastly, we do a little bit of cleanup in the `finalize` block of our `configset` and add the PowerShell-ISE feature to our Windows server to have that ready for any future scripting needs when the deployed instance is in production. We then signal success to the `WaitHandle` resource so the script can take the next logical step and start creating the second Active Directory domain controller (DC2).

Installing and configuring the second domain controller, which we deploy into a different Availability Zone, follows much the same blueprint as we established with the first domain controller. The only difference is that, for the first time, we need to provide a script to join the server to the domain first before promoting it to a replica domain controller. Joining a domain is done as follows (this little building block is used repeatedly in the other scripts):

1. We have to configure the network interface (NIC) on the newly launched server to point the first domain controller (DC1) static IP address as the DNS server. Windows provides several ways of achieving this in a scripted manner. We choose to work with the WMI object `Win32_NetworkAdapterConfiguration`. Our Windows PowerShell script is as follows:

```
"c:\cfn\scripts\SetDNSOnNIC.ps1" : {
  "content" : { "Fn::Join" : [ "\n", [
    "param($Gateway1)",
    "$NICs = Get-WMIObject Win32_NetworkAdapterConfiguration |",
    "where{$_ .IPEnabled -eq \"TRUE\"}", "\n",
    "Foreach($NIC in $NICs) {$DNSServers = \"$Gateway1\", "\n",
    "$NIC.SetDNSServerSearchOrder($DNSServers)", "\n",
    "$NIC.SetDynamicDNSRegistration(\"TRUE\")", "\n" ] ]
  },
}
```

2. When we execute this script in our `join` section of the `configset`, we provide the private IP address of the DC1 as a reference to the template parameter. This code is as follows:

```

"join" : {
  "commands" : {
    "1-set-dns-on-nic" : {
      "command" : { "Fn::Join" : [ "", [
        "powershell.exe ", "-ExecutionPolicy", "
        Unrestricted", " c:\\cfn\\scripts\\SetDNSOnNIC.ps1 ",
        { "Ref" : "AD1PrivateIp" }
      ] ]
    }
  },
},

```

3. After this is set, we can join the domain.

```

"2-join-domain" : {
  "command" : { "Fn::Join" : [ "", [
    "NETDOM join localhost /Domain:", { "Ref" : "DomainDNSName" }, " /user:",
    { "Ref" : "DomainAdminUser" },
    " /password:",
    { "Ref" : "DomainAdminPassword" },
    " /reboot" ] ]
  },
  "waitAfterCompletion" : "forever"
},

```

Now we can move on to promote the server to domain controller.

DNS and DHCP Setup

Before we finalize our setup, deploy our RD Gateway and configure the security groups that will protect our instances and shrink any potential attack surface, we want to spend a little time looking at the options we have available to provide DNS and DHCP services within a VPC.

For domain name services (DNS), we already made the choice and deployed the DNS Server role on our two Active Directory servers. Those DNS servers will hold the “A” records of all the instances we have deployed in our VPC. What about DHCP for dynamically configuring our devices in the VPC? We could configure the DHCP server role on our Active Directory servers but why add load to these servers when we can use a service that is available at no additional charge with every VPC deployed on AWS? For this deployment, we take advantage of the [DHCP Option Set](#) and configure it accordingly. Configuration is, like with many other resource types, a two-step process.

1. First, we define our DHCP option set and the code is as follows:

```

"ContosoDhcpOptions" : {
  "Type" : "AWS::EC2::DHCPOptions",
  "Properties" : {
    "DomainName" : { "Ref" : "DomainDNSName" },
    "DomainNameServers" : ["AmazonProvidedDNS"],
    "NtpServers" : [ { "Ref" : "AD1PrivateIp" } ],
    "NetbiosNameServers" : [ { "Ref" : "AD1PrivateIp" }, { "Ref" :
    "AD2PrivateIp" } ],
    "NetbiosNodeType" : "2",
    "Tags" : [
      { "Key" : "Domain", "Value" : { "Ref" : "DomainDNSName" } } ]
  }
},

```

2. Then we associate our DHCP option set with the VPC.

```

"ContosoVPCDHCPOptionsAssociation" : {
  "Type" : "AWS::EC2::VPCDHCPOptionsAssociation",
  "Properties" : {
    "VpcId" : { "Ref" : "VPC" },
    "DhcpOptionsId" : { "Ref" : "ContosoDhcpOptions" }
  }
}

```

```
    }
  },
```

This is all we have to do to achieve dynamic host configuration and proper NetBIOS name resolution in our VPC.

Setting up Secure Administrative Access using Remote Desktop Gateway

Before performing the final step in scripting the deployment of our infrastructure, setting up a set of security groups designed to isolate the different tiers of our Server Farm, and thereby significantly reducing any attack surface, we want to deploy our RD Gateway. As we designed our architecture for high availability, we deploy two RD Gateways, one in each Availability Zone. This way, we allow access to the resources that may have failed over to the other Availability Zone in case of an Availability Zone outage. For more information on the architectural considerations for deploying RD Gateways for remote server administration, see the “Remote Server Administration” section in the [Securing the Microsoft Platform on Amazon Web Services](#) whitepaper.

Deploying the RD Gateways using AWS CloudFormation in your scripted deployment is straightforward. All concepts that apply for this type of instance deployment were discussed earlier in this article. We launch the instance, join the instance to the domain, and then execute a Windows PowerShell script that installs the RD Gateway feature. The Windows PowerShell script to install a Windows feature is as follows:

```
"C:\\cfn\\scripts\\RDGW.ps1" : {
  "content" : { "Fn::Join" : [ "", [
    "import-module servermanager", "\n",
    "add-windowsfeature RDS-Gateway -IncludeAllSubFeature", "\n"
  ] ] }
},
```

For more information on RD Gateway configuration guidance, read the Microsoft procedure “[Deploying Remote Desktop Gateway Step-by-Step Guide](#)”.

Securing our Infrastructure with Security Groups

In the [Securing the Microsoft Platform on Amazon Web Services](#) and the [SharePoint on AWS: Reference Architecture](#) whitepapers, we discussed in detail the different methods for securing your AWS infrastructure. In this section, we examine how we actually encode our security groups so we repeatedly and reliably deploy an environment that is secure without having to perform any manual steps using the AWS Management Console.

A security group is just another resource of the type [AWS::EC2::SecurityGroup](#) that we declare and that has a set of properties. The two key properties that define how the security group behaves and what kind of filtering it performs are the properties **SecurityGroupIngress** and **SecurityGroupEgress**. For example, the security group we construct in our script to enable all domain member servers to communicate with the domain controllers for authentication and DNS lookup are as follows.

```
"DomainMemberSG" : {
  "Type" : "AWS::EC2::SecurityGroup",
  "Properties" : {
    "GroupDescription" : "Domain Members",
    "VpcId" : { "Ref" : "VPC" },
    "SecurityGroupIngress" : [
      {"IpProtocol" : "tcp", "FromPort" : "53", "ToPort" : "53", "CidrIp" : { "Ref" : "PrivSub1CIDR" } },
      {"IpProtocol" : "udp", "FromPort" : "53", "ToPort" : "53", "CidrIp" : { "Ref" : "PrivSub1CIDR" } },
      {"IpProtocol" : "tcp", "FromPort" : "49152", "ToPort" : "65535", "CidrIp" : { "Ref" : "PrivSub1CIDR" } },
      {"IpProtocol" : "udp", "FromPort" : "49152", "ToPort" : "65535", "CidrIp" : { "Ref" : "PrivSub1CIDR" } },
      {"IpProtocol" : "tcp", "FromPort" : "53", "ToPort" : "53", "CidrIp" : { "Ref" : "PrivSub5CIDR" } },
      {"IpProtocol" : "udp", "FromPort" : "53", "ToPort" : "53", "CidrIp" : { "Ref" : "PrivSub5CIDR" } },
      {"IpProtocol" : "tcp", "FromPort" : "49152", "ToPort" : "65535", "CidrIp" : { "Ref" : "PrivSub5CIDR" } },
      {"IpProtocol" : "udp", "FromPort" : "49152", "ToPort" : "65535", "CidrIp" : { "Ref" : "PrivSub5CIDR" } },
      {"IpProtocol" : "tcp", "FromPort" : "3389", "ToPort" : "3389", "CidrIp" : { "Ref" : "DMZ1CIDR" } },
```

```

    {"IpProtocol" : "tcp", "FromPort" : "3389", "ToPort" : "3389", "CidrIp" : { "Ref" : "DMZ2CIDR" } }
  ]
},

```

In our particular implementation, we only define Ingress rules. If we define no egress rule, the default egress rule of allowing all outbound traffic is applied. In our VPC setting, this is fine, as it removes complexity in managing the security groups. Also, it provides a sufficient level of security, as individual instances (or instances of the same type that we deploy into separate subnets) listen only on the defined set of ports from specified IP ranges or members of other security groups. Our domain members listen to the typical high port range and port 53 (DNS) from our two domain controllers, plus they allow 3389 (RDP) traffic from our DMZs which host the RD Gateways.

With the setup of our security groups that govern access between subnets and instances, we can deploy the foundational infrastructure for our public SharePoint server farm. We deploy the last piece of our infrastructure, Elastic Load Balancing, in the final step when we set up the web front-end servers.

At the end of this step, you will have the following resources of our architecture launched:

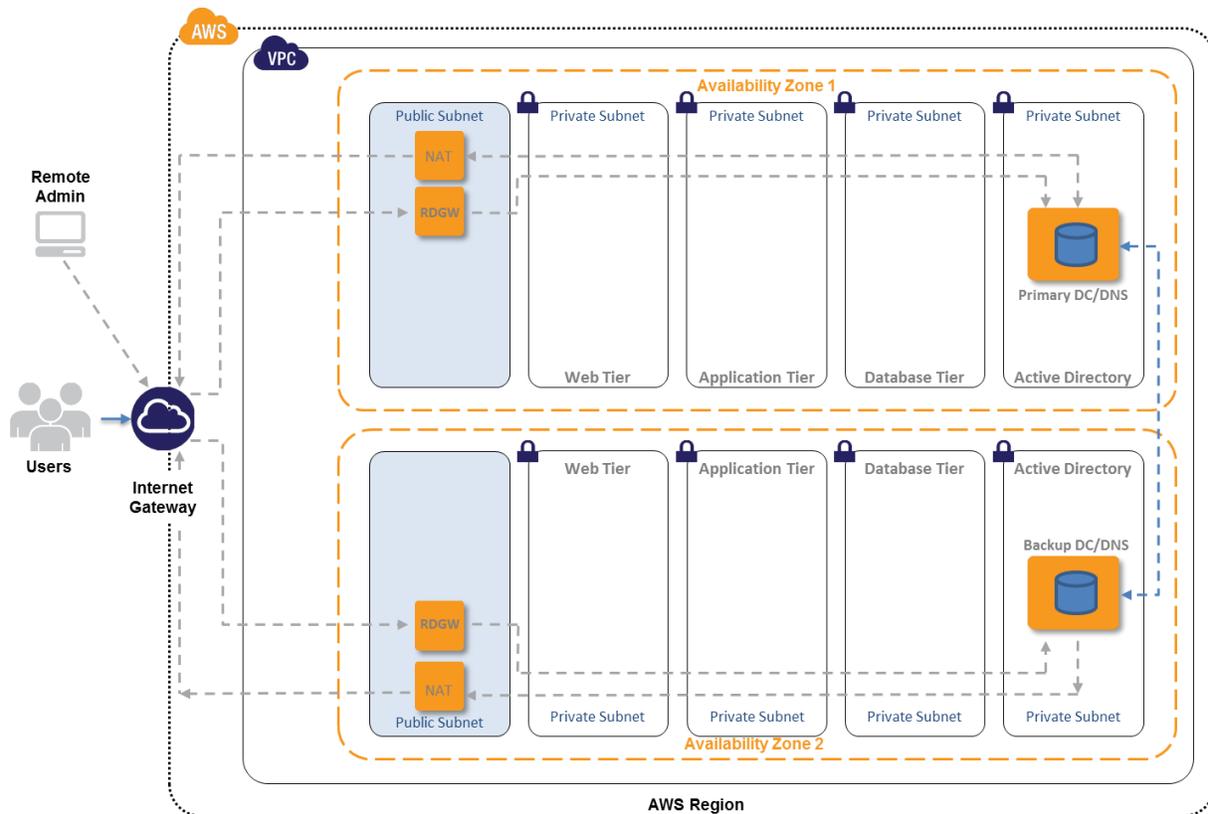


Figure 5: Architecture implemented at the completion of step 2

Step 3: Launch the Database Tier

This step of the advanced implementation guide describes how you can use the AWS CloudFormation service and Windows PowerShell to perform the necessary provisioning and configuration steps to deploy SQL Server 2008 R2 Standard Edition as the database for our database tier, following the recommendations from our [SharePoint Server on AWS: Reference Architecture](#) whitepaper.

NOTE: As with other configuration steps highlighted in this article, this step is meant to be viewed as an example that shows how an administrator might want to script his/her deployment. While the configuration steps performed in our sample template leave you with a SQL Server configuration that is capable of performing well in a medium-size deployment of a SharePoint server farm, you will likely need to make adjustments that reflect the individual circumstances of your deployment.

Let's dive into the individual steps it takes to script the deployment of your SQL Server resources.

Creating a SQL Server 2008 R2 AMI with EC2Config service and AWS CloudFormation Helper Scripts

The current AWS-published SQL Server 2008 R2 Standard Edition AMIs are not AWS CloudFormation enabled. To use a SQL Server 2008 R2 Standard Edition AMI in our scripted infrastructure deployment, we have to extend the existing SQL Server AMI so that it can use the EC2Config Service and the AWS CloudFormation helper scripts. The steps to create such an AMI are as follows:

1. Start with the latest AWS-published SQL Server 2008 R2 Standard Edition AMI.
2. Bring up the instance as a standalone instance (no worries about instance type at this point; anything larger than T1.Micro works).
3. Install the latest [IronPython AWS Cloud Formation tools MSI package](#).
4. Manually reset the UserData flag in C:\Program Files\Amazon\Ec2ConfigService\Settings\Config.xml
 - a. `<Name>Ec2HandleUserData</Name>`
 - b. `<State>Enabled</State>`
5. Manually create the PATH variable in Computer/Properties/Advanced System Settings/Advanced/Environment Variables. Under **System variables** (***NOT*** User Variables), add
 - a. `Path=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Amazon\cfn-bootstrap`
6. Use Sysprep from within the EC2Config service to bundle everything up. Wait until the instance shows up as ***STOPPED*** in the AWS Management Console.
7. Create the AMI image using the AWS Management Console.

Using Sample Template-2

You might want to open up the sample [Template-2](#) AWS CloudFormation template file and follow along.

Template Customization

Sample Template-2 allows for customization of 19 defined parameters at template launch. You can modify those parameters, change the default values, or create an entirely new set of parameters based on your specific deployment scenario. The Template-2 parameters include the following default values.

Parameter	Default	Description
KeyPairName	<User Provides>	Public/private key pairs allow you to connect securely to your instance after it launches.
SQLInstanceType	m2.4xlarge	Amazon EC2 instance type for the SQL Instance
DomainDNSName	contoso.com	Fully qualified domain name (FQDN) of the forest root domain; e.g., corp.example.com.
DomainNetBIOSName	contoso	NetBIOS name of the domain (up to 15 characters) for users of earlier versions of Windows; e.g., CORP.
ServerNetBIOSName	SQL1	NetBIOS name of the SQL Server (up to 15 characters)
DomainAdminUser	StackAdmin	User name for the account that is added as domain administrator. This is separate from the default "administrator" account.
DomainAdminPassword	Password123	Password for the domain admin user. Must be at least 8 characters containing letters, numbers, and symbols.
SPSAdminUser	SPFarmAdmin	User name for the SharePoint server admin account. This account is a domain user and is be added to the SQL Server database as a member of the <i>dbcreator</i> role.
SPSAdminPassword	Password123	Password for the SPS admin user. Must be at least 8 characters containing letters, numbers, and symbols.
AZ1	us-east-1a	Name of Availability Zone that will contain public and private subnets; select a valid zone for your region.
DomainMemberSGID	<User Provides>	ID of the Domain Member Security Group
SqlServerSecurityGroupID	<User Provides>	ID of the SQL Server Security Group
VPC	<User Provides>	ID of the VPC
SQLSvrSubnet	<User Provides>	ID of the Subnet you want to provision the SQL Server into
DataDirectory	D:\SharePoint_Data	Location and Directory of the SharePoint Content and Configuration Databases
LogDirectory	E:\SharePoint_Log	Location and Directory of the SharePoint Database Log Files
SQLAMIID	<User Provides>	ID of the 64-bit CloudFormation enabled SQL AMI available for launch in your region
AD1PrivateIp	10.0.1.10	Fixed private IP for the first Active Directory server located in AZ1
AD2PrivateIp	10.0.5.10	Fixed private IP for the second Active Directory server located in AZ2

Figure 6: Template-2 parameters

Provisioning our SQL Server Instances using AWS CloudFormation and Windows PowerShell

After we have AWS CloudFormation enabled on our SQL Server 2008 R2 Standard Edition AMI, we can use the same methods and some of the building blocks already highlighted in our Step-2 scripting.

Provisioning EBS Volumes

We have looked in the Template-1 sample at ways to assemble Windows PowerShell script on the fly in our AWS CloudFormation template, so let's look—in the context of our SQL Server deployment—at ways for creating batch files.

Our SQL Server instance has, besides the OS EBS volumes, multiple EBS data volumes attached: one for the SharePoint database files and one for the log files. This follows in part Microsoft's recommendations for mitigating any I/O contention. EBS volumes are just another resource, in this case of the type [AWS::EC2::Volume](#). We want to ensure consistent SQL Server I/O performance that is in line with the I/O profile of our application. We therefore create volumes provisioned with the specific number of I/O operations per second (IOPS) that they support. You can attach these provisioned IOPS volumes to special "[EBS-Optimized](#)" instance types. For more information, see the complete list of [all Amazon EC2 instance types](#). Currently, you can launch the following instance types as EBS-Optimized instances.

- Large (m1.large)
- Extra Large (m1.xlarge)
- High-Memory Quadruple Extra Large (m2.4xlarge)

For our sample deployment, we have chosen an m2.4xlarge instance as it is a high memory instance type (68.4 GiB) that lends itself well to a high-performing database server and we attach six additional EBS volumes, each [provisioned with 500 IOPS](#). Out of these six volumes, we create two Raid 0 stripe sets: one with four disks to hold the SharePoint databases, and one with two disks to hold the log files. You can expand on the concepts highlighted in this article and attach additional disks provisioned with higher IOPS, for example, to hold the `tempdbs` database.

In any case, let's start to script the provisioning of our additional volumes. First, we declare our resources as follows:

```
"SQLStripeVolume1" : {
  "Type" : "AWS::EC2::Volume",
  "Properties" : {
    "Size" : "50",
    "Iops" : "500",
    "VolumeType" : "io1",
    "AvailabilityZone" : { "Ref" : "AZ1" }
  }
},
"SQLStripeVolume2" : {
  "Type" : "AWS::EC2::Volume",
  "Properties" : {
    "Size" : "50",
    "Iops" : "500",
    "VolumeType" : "io1",
    "AvailabilityZone" : { "Ref" : "AZ1" }
  }
},
```

Next, we attach these volumes to our instance. You can do this using the **Volumes** property in the Properties section of our instance and the code is as follows:

```
"Properties": {
  "ImageId" : { "Ref" : "SQLAMIID" },
  "InstanceType" : { "Ref" : "SQLInstanceType" },
  "SubnetId" : { "Ref" : "SQLSvrSubnet" },
  "EbsOptimized" : "true",
  "Tags" : [ {
    "Key" : "Name",
    "Value" : { "Ref" : "ServerNetBIOSName" }
  } ],
  "Volumes" : [
    {
      "VolumeId" : { "Ref" : "SQLStripeVolume1" },
      "Device" : "/dev/xvdf"
    },
    {

```

```

"VolumeId" : { "Ref" : "SQLStripeVolume2" },
"Device" : "/dev/xvdg"
},

```

Then we create the batch file and the associated text file to run the DISKPART utility and this is as follows:

```

"C:\\cfn\\scripts\\StripeDisk1.txt" : {
  "content" : { "Fn::Join" : [ "", [
    "select disk=1 ", "\n",
    "select partition 1 ", "\n",
    "delete partition ", "\n",
    "select disk=2 ", "\n",
    "select partition 1 ", "\n",
    "delete partition ", "\n",
    "select disk=3 ", "\n",
    "select partition 1 ", "\n",
    "delete partition ", "\n",
    "select disk=4 ", "\n",
    "select partition 1 ", "\n",
    "delete partition ", "\n",
    "select disk=1 ", "\n",
    "convert dynamic ", "\n",
    "select disk=2 ", "\n",
    "convert dynamic ", "\n",
    "select disk=3 ", "\n",
    "convert dynamic ", "\n",
    "select disk=4 ", "\n",
    "convert dynamic ", "\n",
    "create volume stripe disk=1,2,3,4 ", "\n",
    "select volume=1 ", "\n",
    "assign letter=D ", "\n",
    "format fs=ntfs quick ", "\n",
    "exit", "\n"
  ] ] }
},
"C:\\cfn\\scripts\\StripeDisk1.bat" : {
  "content" : { "Fn::Join" : [ "", [
    "diskpart /s C:\\cfn\\scripts\\StripeDisk1.txt", "\n"
  ] ] }
},

```

This leaves us with subsequently executing the command to run the batch file and is as follows:

```

"createraid" : {
  "commands" : {
    "1-perform-StripeDisk1" : {
      "command" : { "Fn::Join" : [ "", [
        "C:\\cfn\\scripts\\StripeDisk1.bat"
      ] ] }
    },
    "waitAfterCompletion" : "0"
  },
  "2-perform-StripeDisk2" : {
    "command" : { "Fn::Join" : [ "", [
      "C:\\cfn\\scripts\\StripeDisk2.bat"
    ] ] }
    },
    "waitAfterCompletion" : "0"
  }
}
},

```

And that's all there is to automatically provisioning your SQL Server 2008 R2 Standard Edition instance with two Raid 0 stripe sets on an [EBS-Optimized instance](#).

Provision and Configure SQL Server for Use with SharePoint 2010

After having provisioned the necessary volumes and joined the SQL Server instance to the domain (which follows the same steps and re-uses the code highlighted in STEP-2), we want to provision the SQL Server instance so that it:

1. Makes use of the provisioned volumes
2. Allows for a seamless SharePoint configuration experience by adding the SharePoint farm administrator account to specific roles.

You use Windows PowerShell for both of these tasks.

Changing the default database location

As discussed in the previous section, we have provisioned two striped EBS volumes, one to hold the SharePoint databases and one to hold the log files.

[SQL Server Management Objects \(SMO\)](#) help us with accomplishing the task of changing the default database location. We also want to give the users of our scripts some control over naming of the directories that hold the databases and log files. Therefore, we allow the passing of parameters from the AWS CloudFormation template to the Windows PowerShell script. The Windows PowerShell script is as follows:

```
param (
    [string]$datalocation,
    [string]$loglocation
)

#####
# Load the SMO assembly #
#####
[System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO') | out-null

#####
# Create the Directories that hold the Data and Log Files #
#####
md $datalocation
md $loglocation

#####
# Connect to the instance using SMO #
#####
$smo = New-Object Microsoft.SqlServer.Management.Smo.Server -argumentList $_;

#####
# Set the new Data and Log File Location #
#####
$settings = $smo.Settings;
$settings.DefaultFile=$datalocation;
$settings.DefaultLog=$loglocation;
$settings.Alter();

# Stop SQL Server Service
net stop mssqlserver

# Start SQL Server Service
net start mssqlserver
```

You invoke this script and pass the parameters *\$datalocation* and *\$loglocation* as follows, in our AWS CloudFormation script:

```
"4-execute-powershell-script-ChangeDefaultDBandLog" : {
  "command" : { "Fn::Join" : [ "", [
    "powershell.exe ", "-ExecutionPolicy", " Unrestricted", "
    C:\\cfn\\scripts\\ChangeDefaultDBandLog.ps1 -datalocation ", { "Ref" :
    "DataDirectory" }, " -loglocation ", { "Ref" : "LogDirectory" }
  ] ] },
  "waitAfterCompletion" : "0"
},
```

Adding SharePoint Logins to Database roles

To enable the SharePoint farm administrator to perform the farm configuration tasks seamlessly without having to do any additional SQL Server configuration, we also need to add the farm administrator to SQL Server with a login, and associate that login to the role of *dbcreator* and *securityadmin*. The methods we use for downloading the script from our Amazon S3 bucket and running the script is the same as we have used on several other locations; we skip examining these steps in detail.

Opening SQL Server Ports in the Windows Firewall

Besides opening port 1433 and port 1434 in our SQL Server security group, we also need to open those ports on the Windows firewall on the Windows instance itself. For that purpose, we construct a small batch file that calls the *netsh advfirewall* command and performs the desired action. This batch file is as follows:

```
@echo ===== SQL Server Ports =====
@echo Enabling SQLServer default instance port 1433
netsh advfirewall firewall add rule name="SQL Server" dir=in action=allow protocol=TCP localport=1433
@echo Enabling Dedicated Admin Connection port 1434
netsh advfirewall firewall add rule name="SQL Admin Connection" dir=in action=allow protocol=TCP localport=1434
@echo Enabling conventional SQL Server Service Broker port 4022
netsh advfirewall firewall add rule name="SQL Service Broker" dir=in action=allow protocol=TCP localport=4022
@echo Enabling Transact-SQL Debugger/RPC port 135
netsh advfirewall firewall add rule name="SQL Debugger/RPC" dir=in action=allow protocol=TCP localport=135
```

This wraps up the provisioning of our SQL Server instance so it performs well, and it enables the SharePoint farm administrator to perform any SharePoint configuration task without any additional SQL Server configuration.

Configuring SQL Server for High Availability

As discussed in more detail in the [SharePoint on AWS: Reference Architecture](#) whitepaper, you should provision your SQL Server instance for high availability. Running the script we constructed in Step 3, deploying a replica of the primary SQL Server instance as secondary, and mirroring into a different subnet that resides in another Availability Zone takes care of half the steps necessary to set up a SQL Server mirroring solution. All that's left is to set up a witness server and later specify in SharePoint central administration the mirror server name in the **Failover Database** field.

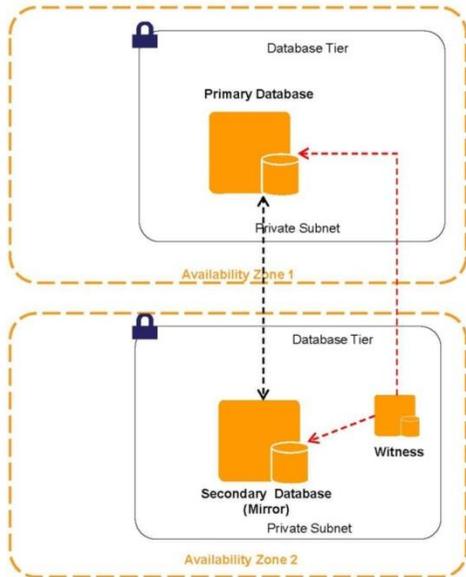


Figure 7: Configuring SQL Server for High Availability

At the end of this step, you will have the following resources of our architecture launched:

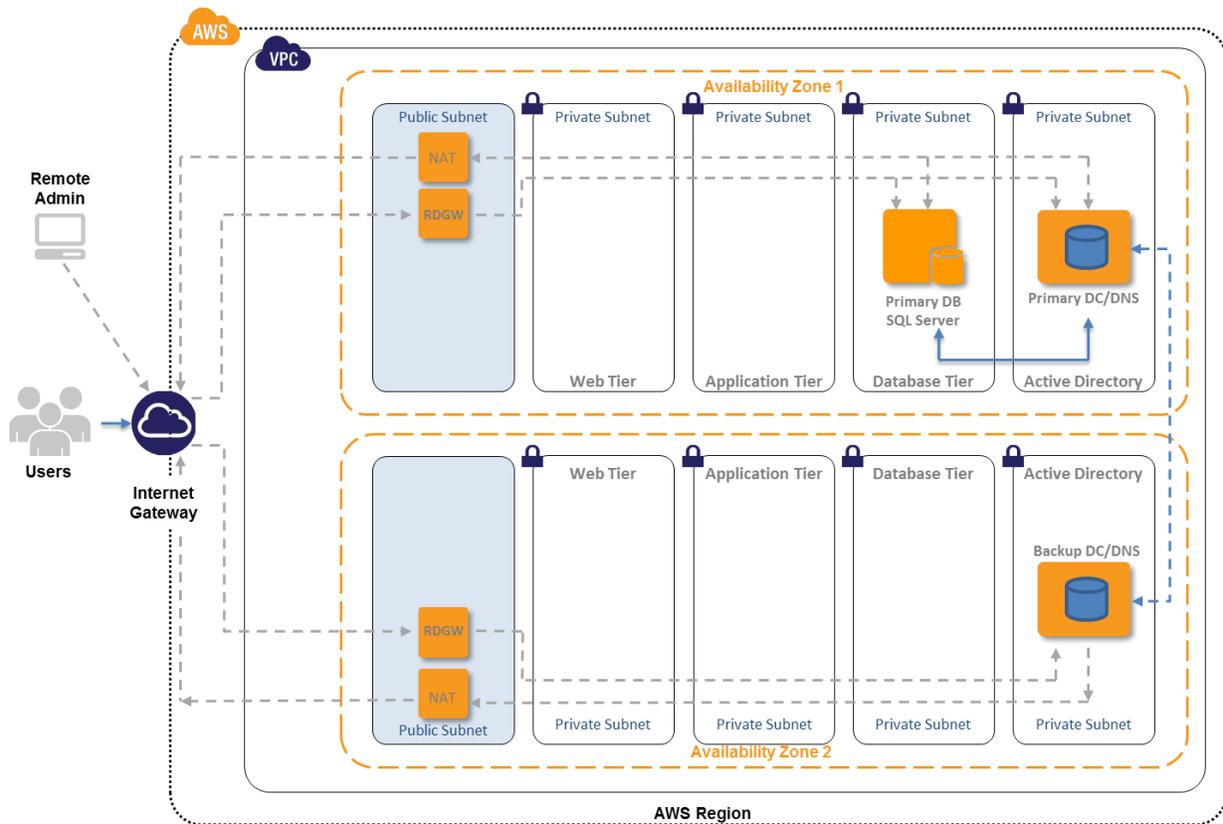


Figure 8: Architecture implemented at the completion of step 3

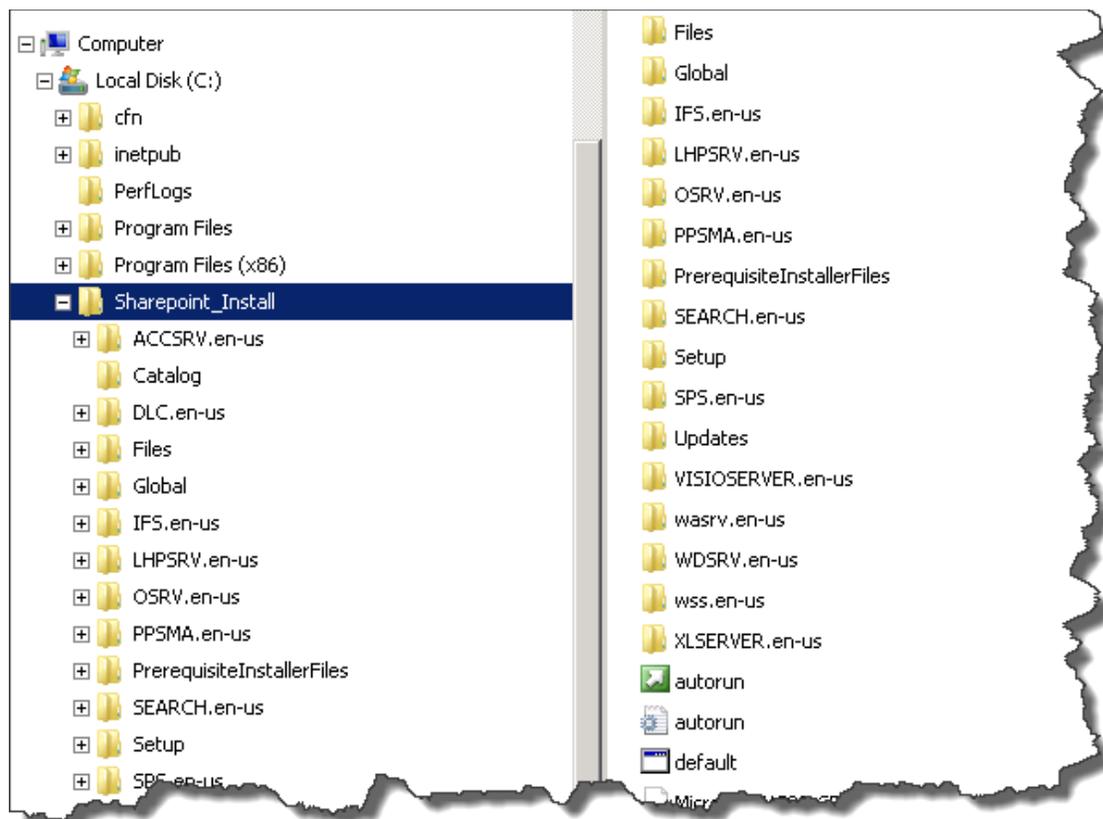
Step 4: Launch the Application (APP) server tier

SharePoint 2010 is not pre-installed in any publicly available AMI. If we want to install SharePoint using the SharePoint deployment scripts so it fits with our architecture and deployed infrastructure, we have to create our own private SharePoint 2010 AMI that holds the raw (uninstalled) SharePoint 2010 bits and the SharePoint Windows PowerShell module (SPModule). In addition, you must obtain sufficient licensing for deploying SharePoint 2010 in AWS (through [Microsoft License Mobility](#)). Our accompanying sample template shows and implements a method for providing your own mobilized license at installation time.

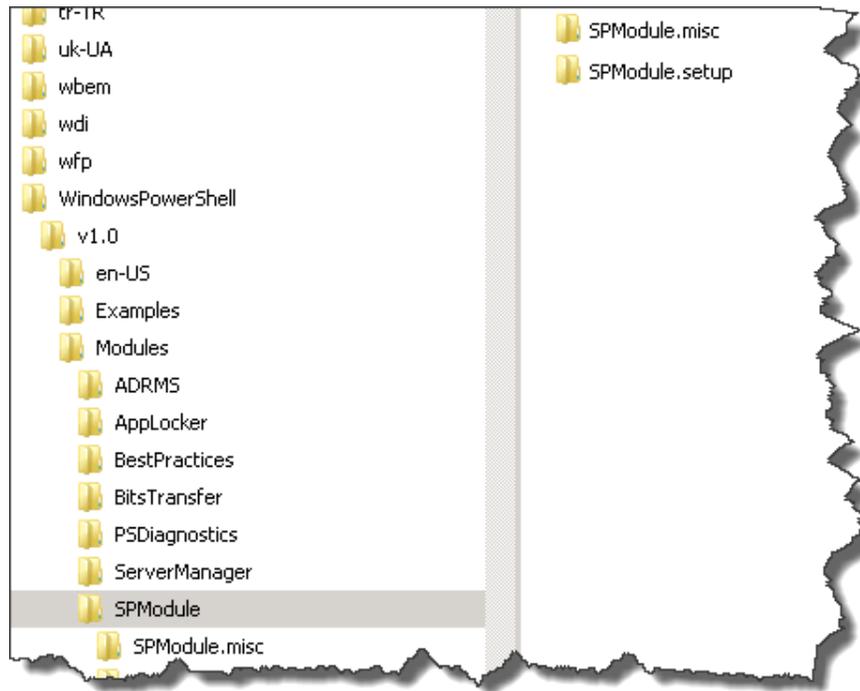
Creating a SharePoint Installation AMI

For both the application Server tier and the web front-end servers, we need to create our own Windows Server AMI that holds the raw (uninstalled) SharePoint bits and the SPModule. The steps for creating such an AMI are as follows:

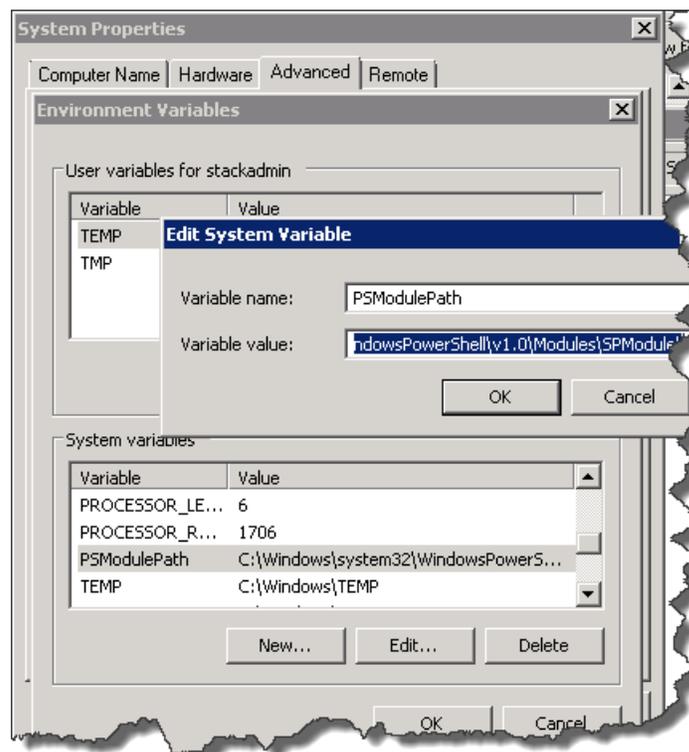
1. Start with the latest AWS-published [Windows Server 2008 R2 English 64-bit - Base for CloudFormation](#) AMI.
2. Bring up the instance as a standalone instance (no worries about instance type at this point, anything larger than T1.Micro works).
3. On the launched instance, download a trial version of SharePoint 2010 for Internet Sites, Enterprise from [sharepoint.microsoft.com](#) (or use your licensed media) and extract the raw (uninstalled) bits to the directory `c:\Sharepoint_install`.



- On the launched instance, download the SPMModule.zip file from the [Microsoft Download Center](#) and extract it to C:\Windows\System32\WindowsPowerShell\v1.0\Modules\SPModule.



- Create or edit the PSModulePath system variable to point to the location where you copied the files in Step 4.



6. After that, manually reset the UserData flag in C:\Program Files\Amazon\Ec2ConfigService\Settings\Config.xml
 - a. <Name>Ec2HandleUserData</Name>
 - b. <State>**Enabled**</State>
7. Use Sysprep from within the EC2Config app to bundle everything up. Wait until the instance shows up as ***STOPPED*** in the AWS Management Console.
8. Create the AMI image using the AWS Management Console.

Using Sample Template-3

You might want to open up the sample [Template-3](#) AWS CloudFormation template file and follow along.

Template Customization

Sample Template-3 allows for customization of 19 defined parameters at template launch. You can modify those parameters, change the default values, or create an entirely new set of parameters based on your specific deployment scenario. The Template-3 parameters include the following default values.

Parameter	Default	Description
KeyPairName	<User Provides>	Public/private key pairs allow you to connect securely to your instance after it launches.
AppSVR1InstanceType	m1.xlarge	Amazon EC2 instance type for the APP Server Instance
DomainDNSName	contoso.com	Fully qualified domain name (FQDN) of the forest root domain; e.g., corp.example.com.
DomainNetBIOSName	contoso	NetBIOS name of the domain (up to 15 characters) for users of earlier versions of Windows; e.g., CORP.
ServerNetBIOSName	APP1	NetBIOS name of the APP Server (up to 15 characters)
DomainAdminUser	StackAdmin	User name for the account that is added as domain administrator. This is separate from the default "administrator" account.
DomainAdminPassword	Password123	Password for the domain admin user. Must be at least 8 characters containing letters, numbers, and symbols.
SPSAdminUser	SPFarmAdmin	User name for the SharePoint server admin account. This account is a domain user and is added to the SQL Server database as a member of the <i>dbcreator</i> role.
SPSAdminPassword	Password123	Password for the SPS admin user. Must be at least 8 characters containing letters, numbers, and symbols.
AZ1	us-east-1a	Name of Availability Zone that will contain public and private subnets; select a valid zone for your region.
AZ2	us-east-1b	Name of Availability Zone that will contain public & private subnets - Select a valid Zone for your region
DomainMemberSGID	<User Provides>	ID of the Domain Member Security Group
AppServerSecurityGroupID	<User Provides>	ID of the App Server Security Group
VPC	<User Provides>	ID of the VPC
AppSvrSubnet	<User Provides>	ID of the APP Server Subnet
SPLicensekey	<User Provides>	A valid SharePoint PID in the format VK7BD-VBKWR-6FHD9-Q3HM9-6PKMX. You can download a SharePoint 2010 trial PID from http://www.microsoft.com/en-us/download/details.aspx?id=16631
SPSAMIID	<User Provides>	ID of the 64-bit CloudFormation enabled SharePoint Server AMI available for launch in your region
AD1PrivateIp	10.0.1.10	Fixed private IP for the first Active Directory server located in AZ1
AD2PrivateIp	10.0.5.10	Fixed private IP for the second Active Directory server located in AZ2

Figure 9: Template-3 parameters

Provisioning the SharePoint Application Server using AWS CloudFormation and Windows PowerShell

First, we provision the additional EBS volumes onto which we install SharePoint. This follows exactly the same blueprint as we have used for provisioning the volumes for our SQL Server instance. However, for the application server, we accept the default EBS volume configuration and provision neither EBS-Optimized instances nor EBS volumes with a specified number of IOPS. (The performance profile of your individual deployment may be different and may require launching EBS-Optimized instances and provisioning a certain number of IOPS.)

We join the instance to the domain and then we are ready to install SharePoint.

Installing SharePoint with a License or PID Key

Now we are ready to install the SharePoint bits on our application server using our own license key or, for a trial or test installation, a Trial PID Key.

Installing SharePoint on our instance in a scripted fashion follows the scripted deployment reference as documented in the [Scripted deployment reference \(SharePoint Server 2010\)](#) topic. The short Windows PowerShell script that executes the Install-SharePoint cmdlet provides for a parameter called *\$productkey* to be passed in. We provide this key from a parameter in our AWS CloudFormation script.

```
#####
# Script Parameters #
#####
param (
    [string]$productkey
)
#####
# Ensure I got the right environment variable to the Module Path in my session #
#####
$env:PSModulePath = $env:PSModulePath + ";C:\Windows\System32\WindowsPowerShell\v1.0\Modules\SPModule"

#####
# Import the SharePoint Powershell modules #
#####
import-module spmodule.misc
import-module spmodule.setup

#####
# Install Central Admin #
#####
Install-SharePoint -SetupExePath "c:\SharePoint_Install\setup.exe" -PIDKey $productkey -InstallDirectory
"d:\SharePoint" -DataDirectory "d:\SharePoint\Data"
```

Calling this script from AWS CloudFormation is as follows:

```
"4-execute-powershell-script-SPInst" : {
  "command" : { "Fn::Join" : [ "", [
    "powershell.exe ", "-ExecutionPolicy", " Bypass", " C:\\cfn\\scripts\\SPInst.ps1 -productkey \"", { "Ref" : "SPLicensekey" }, "
  ] ] },
  "WaitAfterCompletion" : "0"
}
```

Note the reference to the SPLicensekey { "Ref" : "SPLicensekey" }. You will find this when launching the template. Please also note that you are responsible for complying with Microsoft's requirements for your use of the SharePoint 2010 trial. Microsoft may discontinue or change the requirements around the SharePoint 2010 trial at any time. If you have questions about Microsoft's requirements or the SharePoint 2010 trial, you may be able to find answers at Microsoft's SharePoint evaluation site: <http://technet.microsoft.com/en-us/library/cc261970.aspx>. For more information on how

you can provide your mobilized SharePoint license into the deployment process, see Microsoft's [License Mobility through Software Assurance](#) program.

Specify Parameters

Below are the parameters associated with your CloudFormation template. You may review and proceed with the default parameters or make customizations as needed below.

AppServerSecurityGroup
ID of the App Server Security Group

SPSAdminPassword
Password for the SPS admin user. Must be at least 8 characters containing letters, numbers and symbols

SPLicensekey
A valid SharePoint PID in the format VK7BD-VBKWR-6FHD9-Q3HM9-6PKMX. You can download a SharePoint 2010 trial and trial PID from <http://www.microsoft.com/en-us/download/details.aspx?id=16631>

AppSVR1InstanceType

With that and just a few lines of AWS CloudFormation and Windows PowerShell, we have finished installing SharePoint on our application server. We use the same script to deploy more than one application server, when we want to either create application server groups in the same Availability Zone or spread the installation out across Availability Zones. We change the NetBIOS name of our instance and the subnet ID if we want to deploy into a different Availability Zone.

At the end of this step, you will have the following resources of our architecture launched:

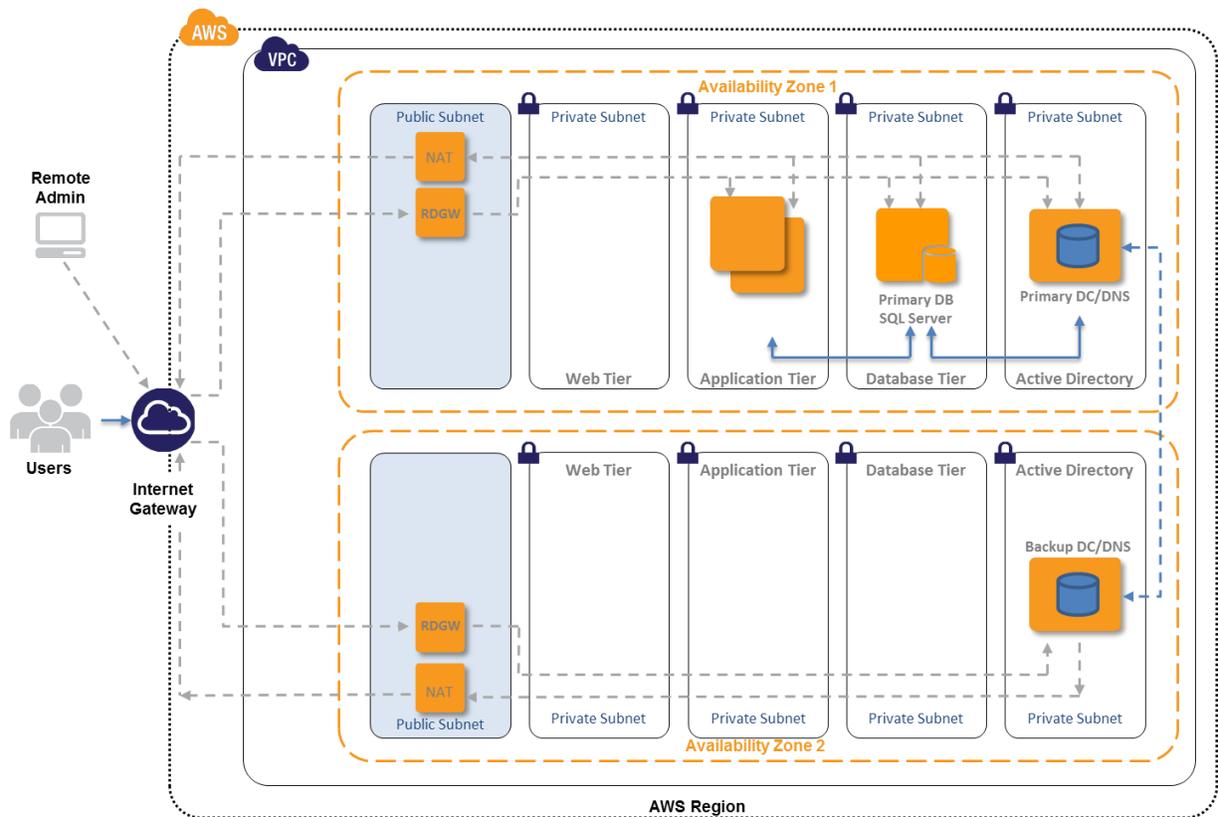


Figure 10: Architecture implemented at the completion of step 4

Step 5: Launch the Web Front-End (WFE) Tier

We have made it all the way to the final step, deploying and provisioning our WFE servers and providing—using Elastic Load Balancing—the only ingress into our VPC from the Internet (besides ingress on RDP port 3389 using the RD Gateway for administrative purposes).

By now, we have executed every module we have scripted for this deployment several times so at this stage you will be familiar with the accompanying script. We are using the same private AMI we have created for the application server instances; we provision a volume, join the domain, and execute the installation Windows PowerShell script with a user-provided license. In the sample script, we deploy two WFE servers so we can demonstrate the effectiveness of Elastic Load Balancing.

Using Sample Template-4

You might want to open up the sample [Template-4](#) AWS CloudFormation template file and follow along.

Template Customization

Sample Template-4 allows for customization of 24 defined parameters at template launch. You can modify those parameters, change the default values, or create an entirely new set of parameters based on your specific deployment scenario. The Template-4 parameters include the following default values.

Parameter	Default	Description
KeyPairName	<User Provides>	Public/private key pairs allow you to connect securely to your instance after it launches.
WebSVR1InstanceType	m1.xlarge	Amazon EC2 instance type for the WFE Server Instance
DomainDNSName	contoso.com	Fully qualified domain name (FQDN) of the forest root domain; e.g., corp.example.com.
DomainNetBIOSName	contoso	NetBIOS name of the domain (up to 15 characters) for users of earlier versions of Windows; e.g., CORP.
Server1NetBIOSName	WFE1	NetBIOS name of the 1st WFE Server (up to 15 characters)
Server2NetBIOSName	WFE2	NetBIOS name of the 2nd WFE Server (up to 15 characters)
DomainAdminUser	StackAdmin	User name for the account that is added as domain administrator. This is separate from the default "administrator" account.
DomainAdminPassword	Password123	Password for the domain admin user. Must be at least 8 characters containing letters, numbers, and symbols.
SPSAdminUser	SPFarmAdmin	User name for the SharePoint server admin account. This account is a domain user and is be added to the SQL Server database as a member of the <i>dbcreator</i> and <i>securityadmin</i> role.
SPSAdminPassword	Password123	Password for the SPS admin user. Must be at least 8 characters containing letters, numbers, and symbols.
AZ1	us-east-1a	Name of Availability Zone that will contain public and private subnets; select a valid zone for your region.
AZ2	us-east-1b	Name of Availability Zone that will contain public & private subnets - Select a valid Zone for your region
DomainMemberSGID	<User Provides>	ID of the Domain Member Security Group
WfeServerSecurityGroupID	<User Provides>	ID of the Web Front End Server Security Group
LBSecurityGroupID	<User Provides>	ID of the Load Balancer Security Group
VPC	<User Provides>	ID of the VPC
WFESvrSubnet	<User Provides>	ID of the WFE1 Server Subnet
WFE2SvrSubnet	<User Provides>	ID of the WFE2 Server Subnet
DMZ1Subnet	<User Provides>	ID of the DMZ1 Subnet
DMZ2Subnet	<User Provides>	ID of the DMZ2 Subnet
SPLicensekey	<User Provides>	A valid SharePoint PID in the format of VK7BD-VBKWR-6FHD9-Q3HM9-6PKMX. You can download a SharePoint 2010 trial PID from http://www.microsoft.com/en-us/download/details.aspx?id=16631
SPSAMIID	<User Provides>	ID of the 64-bit CloudFormation enabled SharePoint Server AMI available for launch in your region
AD1PrivateIp	10.0.1.10	Fixed private IP for the first Active Directory server located in AZ1
AD2PrivateIp	10.0.5.10	Fixed private IP for the second Active Directory server located in AZ2

Figure 11: Template-4 parameters

Adding Elastic Load Balancing

As with all other resources we have deployed in the preceding steps, Elastic Load Balancing is just that—yet another resource. This time, it's a resource of the type [AWS::ElasticLoadBalancing::LoadBalancer](#). The script for launching and configuring Elastic Load Balancing is as follows:

```
"ElasticLoadBalancer" : {
  "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
  "Properties" : {
    "Subnets" : [ { "Ref" : "DMZ1Subnet" }, { "Ref" : "DMZ2Subnet" } ],
    "Instances" : [ { "Ref" : "WebSVR1" }, { "Ref" : "WebSVR2" } ],
    "Listeners" : [ {
      "LoadBalancerPort" : "80",
      "InstancePort" : "80",
      "Protocol" : "HTTP"
    } ],
    "SecurityGroups" : [ { "Ref" : "LBSecurityGroupID" } ],
    "HealthCheck" : {
      "Target" : "TCP:80",
      "HealthyThreshold" : "3",
      "UnhealthyThreshold" : "5",
      "Interval" : "10",
      "Timeout" : "5"
    }
  }
},
```

Let's take a closer look at what we are configuring here and why. First, the "Subnets" properties define into which subnets we actually deploy the LoadBalancers. While there is only one routable IP address for the LoadBalancer, we deploy individual instances of the LoadBalancer in the defined subnets. As this is a public LoadBalancer and not a VPC-internal LoadBalancer, the only subnets we can deploy Elastic Load Balancing into in our setup are the DMZ subnets, as they have the public route using the Internet gateway configured.

Secondly, the "Instances" properties define which individual instances we actually want to hook up to the ELB. In our case, this is, of course, the two WFE servers.

The "Listeners" properties define that Elastic Load Balancing is listening on and only accepts HTTP traffic on Port 80 and so are the WFE instances. This is also enforced using the LoadBalancer security group, which only allows ingress on port 80 from the Internet, plus the WFE security group, which specifies that ingress is only allowed on port 80 from the LoadBalancer security group.

With that, the provisioning and configuration of the necessary infrastructure to run a public website on AWS is complete. After configuring your individual SharePoint farm and web applications (or follow the simple installation steps in the appendix to set up a default team site), you will be able to connect to them using the IP address of LoadBalancer.

At the end of this step, you will have the following resources of our architecture launched:

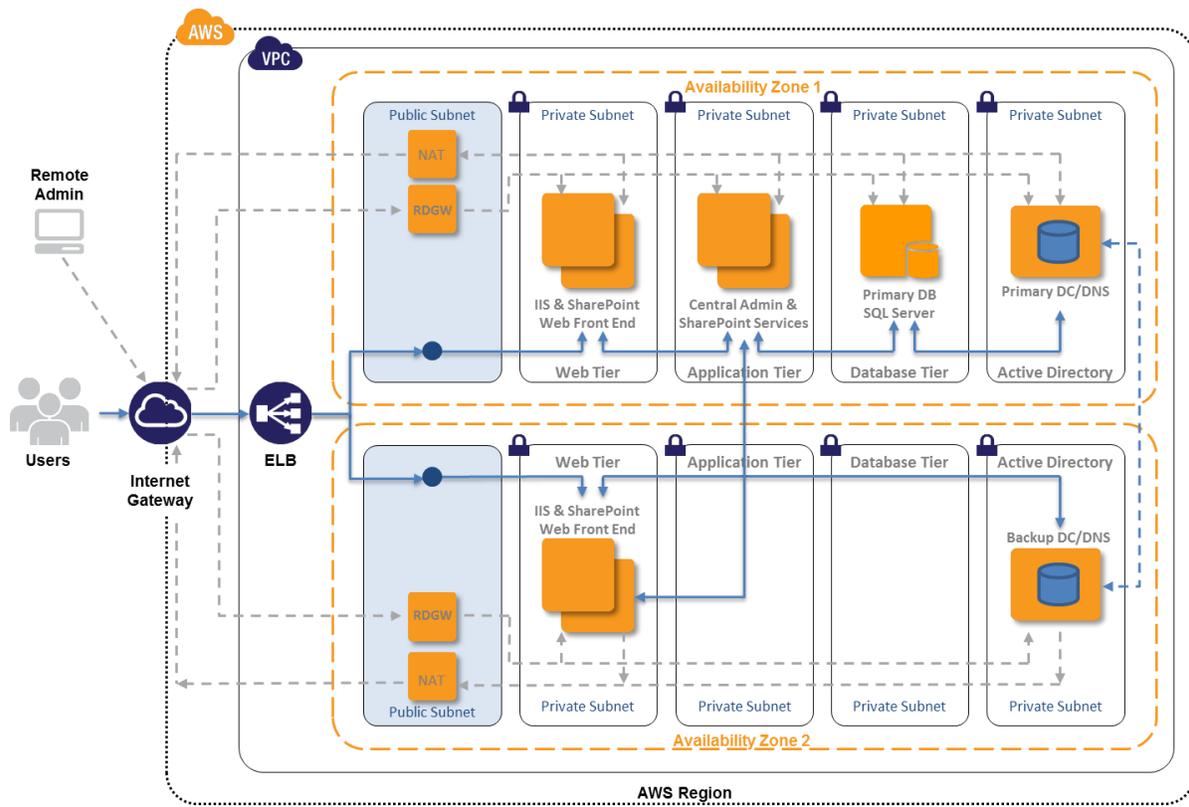


Figure 12: Architecture implemented at the completion of step 5

Step 6: Configure the SharePoint Server Farm

After you have launched the complete stack, you need to configure your SharePoint Farm. It is outside the scope of this article to discuss the many configuration options that SharePoint supports. However, for the purpose of creating a simple proof of concept (POC) or demonstration setup, you will need to perform the following configuration steps:

On the APP Server:

- Run the SharePoint Products and Technologies Configuration Wizard.
- Create a new server farm.

On the WFE1 Server:

- Run the SharePoint Products and Technologies Configuration Wizard.
- Connect to an existing farm.
- Launch the Farm Configuration Wizard.
- Create a Site Collection.

On the WFE2 Server:

- Run the SharePoint Products and Technologies Configuration Wizard.
- Connect to an existing farm.

Installing and Configuring SharePoint 2010 on APP1

This describes the minimum necessary manual configuration steps if you want to demonstrate the facilities of a default team site.

To install and configure SharePoint 2010 on APP1

1. Log into the APP1 instance using RDP from the RDGW1 instance using <contoso\spfadmin> and password <Password123>.

NOTE: If you have changed any of the default parameters as discussed in Step 2 of this article, you must adjust the user ID and password accordingly.
2. Run the SharePoint Products and Technologies Configuration Wizard.
3. On the **Welcome to SharePoint Products** page, click **Next**.
4. In the message that notifies you that some services might need to be restarted during configuration, click **Yes**.
5. On the **Connect to a server farm** page, click **Create a new server farm**, and then click **Next**.
6. On the **Specify Configuration Database Settings** page, type **SQL1** in **Database server**, type **CONTOSO\SPFarmAdmin** in **User name**, type **Pssword123** in **Password**, and then click **Next**.
7. On the **Specify Farm Security Settings** page, type **Password123** in both **Passphrase** and **Confirm passphrase**, and then click **Next**.
8. On the **Configure SharePoint Central Administration Web Application** page, check the **Specify Port number** box and type port **5000**, and click **Next**.

Note: You can choose to specify another port number or let the SharePoint Configuration Wizard choose one for you. If you do so, you must update the ingress rule on the application server security group accordingly.
9. On the **Completing the SharePoint Products Configuration Wizard** page, click **Next**.
10. On the **Configuration Successful** page, click **Finish**. Internet Explorer launches with a tab named Initial Farm Configuration Wizard.
11. In the **Help Make SharePoint Better** dialog box, click **No, I don't wish to participate**, and then click **OK**.
12. For **How do you want to configure your SharePoint farm?**, click **Start the Wizard**.
13. On the **Configure your SharePoint farm** page, in **Service account**, click **Use existing managed account**, and then click **Next**.
14. On the **Create Site Collection** page, click **Skip**.
15. On the **"This completes the Farm Configuration Wizard"** page, click **Finish**. The Internet Explorer tab shows the SharePoint 2010 Central Administration site, from which you can configure and manage the SharePoint server. Leave Internet Explorer open.

Installing and Configuring SharePoint 2010 on WFE Servers

This procedure describes the minimum necessary manual configuration steps if you want to demonstrate the facilities of a default team site.

To install and configure SharePoint 2010 on WFE1

1. Log into the WFE1 instance using RDP from the RDGW1 instance using <contoso\spfarmadmin> and password <Password123>.
2. Run the SharePoint Products and Technologies Configuration Wizard.
3. On the **Welcome to SharePoint Products** page, click **Next**.
4. In the message that notifies you that some services might need to be restarted during configuration, click **Yes**.
5. On the **Connect to a server farm** page, click **Connect to an existing server farm**, and then click **Next**.
6. On the **Specify Configuration Database Settings** page, type **SQL1** in **Database server**, and then click **Retrieve Database Names**.
7. Click **SharePoint_Config** in the **Database name** list, and then click **Next**.
8. On the **Specify Farm Security Settings** page, type **Password123** in **Passphrase**, and then click **Next**.
9. On the **Completing the SharePoint Products Configuration Wizard** page, click **Next**.
10. On the **Configuration Successful** page, click **Finish**. The Internet Explorer tab shows the SharePoint 2010 Central Administration site. Leave Internet Explorer open.
11. On APP1, in the Internet Explorer window for SharePoint Central Administration, in **System Settings**, click **Manage servers in this farm** and verify that WFE1 is part of the farm.
12. On WFE1, from Internet Explorer and the Central Administration tab, click **Configuration Wizards**, and then click the **Launch the Farm Configuration Wizard**.
13. For **How do you want to configure your SharePoint farm?**, click **Start the Wizard**.
14. On the **Configure your SharePoint farm** page, click **Next**.
15. On the **Create Site Collection** page, in **Title and description**, type **Contoso Corporation** in **Title**, and then click **OK**. This step creates a team site at the URL <http://app1>.
16. On the **This completes the Farm Configuration Wizard** page, click **Finish**. The Internet Explorer tab shows the SharePoint 2010 Central Administration site, from which you can configure and manage the SharePoint server.

To install and configure SharePoint 2010 on WFE2

1. Repeat the steps from Step 1-12 to configure SharePoint 2010 on WFE2 instance.

Testing your SharePoint 2010 Server Deployment and Demonstrating the Facilities of the Default Team Site

In this procedure, you click through the default SharePoint facilities and resources for the Contoso Corporation team site. Connect to the WFE instance using the IP address of the LoadBalancer.

To test the facilities of the default Contoso Corporation team site:

1. In a web browser, connect to the IP address of the LoadBalancer.
2. On the SharePoint team site for the Contoso Corporation, provide your credentials. Enter the credentials of the contoso\spfarmadmin account.
3. Under **Libraries**, click **Site Pages**. These Web pages can act as a team wiki library to share information and ideas and collaborate among the members of your team. You can click **Add new page** to create a new site page. Click **How to Use This Library** to show the Wiki page that tells you how to use this wiki library.
4. Under **Libraries**, click **Shared Documents**. Multiple people can work on each document in a central location, which makes document collaboration much easier than sending files in email. You can click **Add document** to add a new document to the list of shared documents.
5. Under **Lists**, click **Calendar**. You can manage and share events on this calendar across your team. You can point to a day and then click **Add** to add a calendar event.
6. Under **Lists**, click **Tasks**. You can add and edit tasks on this list, and track progress. You can click **Add new item** to add a new task to this list.
7. Under **Discussions**, click **Team Discussion**. This is a simple newsgroup-style team discussion forum from which you can ask questions and share information for issues and topics important to your team. You can click **Add new discussion** to add a new topic for discussion.
8. Click **Home** to return to the home page of the Contoso Corporation team site.

Further Reading

- Microsoft on AWS:
 - <http://aws.amazon.com/microsoft/>
- Amazon EC2 Windows Guide:
 - <http://docs.amazonwebservices.com/AWSEC2/latest/WindowsGuide/Welcome.html?r=7870>
- Microsoft AMIs for Windows and SQL Server:
 - <http://aws.amazon.com/windows>
 - <http://aws.amazon.com/amis/Microsoft?browse=1>
 - <http://aws.amazon.com/amis/6258880392999312> (SQL Server)
- AWS Windows and .NET Developer Center:
 - <http://aws.amazon.com/net>
- Microsoft License Mobility:
 - <http://aws.amazon.com/windows/mslicenseability>
- Whitepapers:
 - Microsoft SharePoint Server on AWS: Reference Architecture – http://awsmedia.s3.amazonaws.com/SharePoint_on_AWS_Reference_Architecture_White_Paper.pdf
 - Using Windows ADFS for Single Sign-On to EC2 - http://media.amazonwebservices.com/EC2_ADFS_howto_2.0.pdf
 - Amazon's Corporate IT Deploys SharePoint 2010 to the Amazon Web Services Cloud - http://media.amazonwebservices.com/AWS_Amazon_SharePoint_Deployment.pdf
 - Relational Database Management Systems in the Cloud: Microsoft SQL Server 2008 R2 - <http://aws.amazon.com/whitepapers/rdbms-in-the-cloud>
 - Providing SSO to Amazon EC2 Apps from an On-premises Windows Domain - <http://download.microsoft.com/download/6/C/2/6C2DBA25-C4D3-474B-8977-E7D296FBFE71/EC2-Windows%20SSO%20v1%20--Chappell.pdf>
 - Secure Microsoft Applications on AWS- http://media.amazonwebservices.com/AWS_Microsoft_Platform_Security.pdf